Studies in Computational Intelligence 380

Anthony Brabazon
Michael O'Neill
Dietmar Maringer (Eds.)

Natural Computing in Computational Finance

Volume 4

Springer

Studies in Computational Intelligence, Volume 380

Editor-in-Chief
Prof. Janusz Kacprzyk
Systems Research Institute
Polish Academy of Sciences
ul. Newelska 6
01-447 Warsaw
Poland
E-mail: kacprzyk@ibspan.waw.pl

Further volumes of this series can be found on our homepage: springer.com

Vol. 356. Slawomir Koziel and Xin-She Yang (Eds.)
Computational Optimization, Methods and Algorithms, 2011
ISBN 978-3-642-20858-4

Vol. 357. Nadia Nedjah, Leandro Santos Coelho, Viviana Cocco Mariani, and Luiza de Macedo Mourelle (Eds.)
Innovative Computing Methods and their Applications to Engineering Problems, 2011
ISBN 978-3-642-20957-4

Vol. 358. Norbert Jankowski, Włodzisław Duch, and Krzysztof Grąbczewski (Eds.)
Meta-Learning in Computational Intelligence, 2011
ISBN 978-3-642-20979-6

Vol. 359. Xin-She Yang, and Slawomir Koziel (Eds.)
Computational Optimization and Applications in Engineering and Industry, 2011
ISBN 978-3-642-20985-7

Vol. 360. Mikhail Moshkov and Beata Zielosko
Combinatorial Machine Learning, 2011
ISBN 978-3-642-20994-9

Vol. 361. Vincenzo Pallotta, Alessandro Soro, and Eloisa Vargiu (Eds.)
Advances in Distributed Agent-Based Retrieval Tools, 2011
ISBN 978-3-642-21383-0

Vol. 362. Pascal Bouvry, Horacio González-Vélez, and Joanna Kolodziej (Eds.)
Intelligent Decision Systems in Large-Scale Distributed Environments, 2011
ISBN 978-3-642-21270-3

Vol. 363. Kishan G. Mehrotra, Chilukuri Mohan, Jae C. Oh, Pramod K. Varshney, and Moonis Ali (Eds.)
Developing Concepts in Applied Intelligence, 2011
ISBN 978-3-642-21331-1

Vol. 364. Roger Lee (Ed.)
Computer and Information Science, 2011
ISBN 978-3-642-21377-9

Vol. 365. Roger Lee (Ed.)
Computers, Networks, Systems, and Industrial Engineering 2011, 2011
ISBN 978-3-642-21374-8

Vol. 366. Mario Köppen, Gerald Schaefer, and Ajith Abraham (Eds.)
Intelligent Computational Optimization in Engineering, 2011
ISBN 978-3-642-21704-3

Vol. 367. Gabriel Luque and Enrique Alba
Parallel Genetic Algorithms, 2011
ISBN 978-3-642-22083-8

Vol. 368. Roger Lee (Ed.)
Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing 2011, 2011
ISBN 978-3-642-22287-0

Vol. 369. Dominik Ryżko, Piotr Gawrysiak, Henryk Rybinski, and Marzena Kryszkiewicz (Eds.)
Emerging Intelligent Technologies in Industry, 2011
ISBN 978-3-642-22731-8

Vol. 370. Alexander Mehler, Kai-Uwe Kühnberger, Henning Lobin, Harald Lüngen, Angelika Storrer, and Andreas Witt (Eds.)
Modeling, Learning, and Processing of Text Technological Data Structures, 2011
ISBN 978-3-642-22612-0

Vol. 371. Leonid Perlovsky, Ross Deming, and Roman Ilin (Eds.)
Emotional Cognitive Neural Algorithms with Engineering Applications, 2011
ISBN 978-3-642-22829-2

Vol. 372. António E. Ruano and Annamária R. Várkonyi-Kóczy (Eds.)
New Advances in Intelligent Signal Processing, 2011
ISBN 978-3-642-11738-1

Vol. 373. Oleg Okun, Giorgio Valentini, and Matteo Re (Eds.)
Ensembles in Machine Learning Applications, 2011
ISBN 978-3-642-22909-1

Vol. 374. Dimitri Plemenos and Georgios Miaoulis (Eds.)
Intelligent Computer Graphics 2011, 2011
ISBN 978-3-642-22906-0

Vol. 375. Marenglen Biba and Fatos Xhafa (Eds.)
Learning Structure and Schemas from Documents, 2011
ISBN 978-3-642-22912-1

Vol. 376. Toyohide Watanabe and Lakhmi C. Jain (Eds.)
Innovations in Intelligent Machines – 2, 2011
ISBN 978-3-642-23189-6

Vol. 377. Roger Lee (Ed.)
Software Engineering Research, Management and Applications 2011, 2011
ISBN 978-3-642-23201-5

Vol. 378. János Fodor, Ryszard Klempous, and Carmen Paz Suárez Araujo (Eds.)
Recent Advances in Intelligent Engineering Systems, 2011
ISBN 978-3-642-23228-2

Vol. 379. Ferrante Neri, Carlos Cotta, and Pablo Moscato (Eds.)
Handbook of Memetic Algorithms, 2011
ISBN 978-3-642-23246-6

Vol. 380. Anthony Brabazon, Michael O'Neill, and Dietmar Maringer (Eds.)
Natural Computing in Computational Finance, 2011
ISBN 978-3-642-23335-7

Anthony Brabazon, Michael O'Neill,
and Dietmar Maringer (Eds.)

Natural Computing in Computational Finance

Volume 4

Springer

Editors

Prof. Anthony Brabazon
Quinn School of Business
University College Dublin
Belfield
Dublin 4
Ireland
E-mail: anthony.brabazon@ucd.ie

Dr. Michael O'Neill
UCD Complex Adaptive Systems Laboratory
University College Dublin
Belfield
Dublin 4
Ireland
E-mail: m.oneill@ucd.ie

Prof. Dietmar Maringer
University of Basel,
Büro 5.56
Wirtschaftswissenschaftliches Zentrum (WWZ)
Abteilung Quantitative Methoden
Peter Merian-Weg 6
4002 Basel
Switzerland
E-mail: dietmar.maringer@unibas.ch

ISBN 978-3-642-23335-7 e-ISBN 978-3-642-23336-4

DOI 10.1007/978-3-642-23336-4

Studies in Computational Intelligence ISSN 1860-949X

Library of Congress Control Number: 2008922057

Typeset & *Cover Design:* Scientific Publishing Services Pvt. Ltd., Chennai, India.

Printed on acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

To Maria
Tony

To Gráinne, Aoife and Michael
Michael

To Klaus
Dietmar

Preface

The field of Natural Computing has been the focus of a substantial research effort in recent decades. One particular strand of this research concerns the development of computational algorithms using metaphorical inspiration from systems and phenomena that occur in the natural world. These natural computing algorithms have proven to be successful problem solvers across domains as diverse as finance, management science, bioinformatics, marketing, engineering and architecture, to name but a few. This edited volume brings together a series of chapters which illustrate the application of a range of cutting-edge natural computing and agent-based methodologies in computational finance and economics. While describing cutting edge applications, the chapters are written so that they are accessible to a wide audience. Hence, they should be of interest to academics, students and practitioners in the fields of computational finance and economics.

The inspiration for this book was due in part to the success of EvoFIN 2010, the 4th European Workshop on Evolutionary Computation in Finance and Economics. EvoFIN 2010 took place in conjunction with Evo* 2010 in Istanbul, Turkey (7-9 April 2010). Evo* is an annual collection of European conferences and workshops broadly focused on Evolutionary Computation. It is the largest European event dedicated to this growing field of research. A number of the chapters presented in this book are extended versions of papers presented at EvoFIN 2010 and these have undergone the same rigorous, peer-reviewed, selection process as the other chapters. This book follows on from previous volumes in this series, namely, **Natural Computing in Computational Finance Volumes I, II and III**.

We would like to thank all the authors for their high-quality contributions and the reviewers who generously gave of their time to peer-review all submissions. We would also like to thank Dr. Thomas Ditzinger of Springer-Verlag and Professor Janusz Kacprzyk, editor of this book series, for their encouragement of, and their support during, the preparation of this book. Finally, Anthony Brabazon and Michael O'Neill would like to acknowledge the support of their research activities provided by Science Foundation Ireland (Grant number 08/SRC/FM1389).

Dublin and Basel
May 2011

Anthony Brabazon
Michael O'Neill
Dietmar Maringer

Contents

1

Natural Computing in Computational Finance (Volume 4): Introduction

Anthony Brabazon[1,2], Michael O'Neill[1,3], and Dietmar Maringer[4]

[1] Natural Computing Research & Applications Group, Complex & Adaptive Systems Laboratory, University College Dublin, Ireland
{anthony.brabazon,m.oneill}@ucd.ie
[2] School of Business, University College Dublin, Ireland
[3] School of Computer Science and Informatics, University College Dublin, Ireland
[4] Business and Economics Faculty, University of Basel, Switzerland
dietmar.maringer@unibas.ch

1.1 Introduction

Natural computing (NC) can be broadly defined as the development of computer programs and computational algorithms using metaphorical inspiration from systems and phenomena that occur in the natural world. The inspiration for natural computing methodologies typically stem from real-world phenomena which exist in high-dimensional, dynamic, environments characteristics which fit well with the nature of financial markets. Prima facie, this makes natural computing methods interesting for financial applications.

Natural Computing algorithms can be clustered into different groups depending on the aspects of the natural world upon which they are based. The main clusters are *Neurocomputing* (which loosely draws its inspiration from the workings of the human brain), *Evolutionary Computing* (which draws inspiration from the processes of biological evolution), *Social Computing* (adopting swarming or communication behaviours of social animal such as birds, fish or insects), *Immunocomputing* (algorithms inspired by biological immune systems) and *Physical Computing* (mimicking chemical or physical processes). Readers interested in a broader introduction to these methods are referred to [1, 2, 3, 5, 9, 11, 13] and the literature quoted therein.

As in the previous volumes of this series, this book covers a variety of financial applications. The book consists of a series of chapters each of which was selected following a rigorous, peer-reviewed, selection process. The chapters illustrate the application of a range of cutting-edge natural computing and agent-based methodologies in computational finance and economics. The applications explored include option model calibration, financial trend reversal detection, enhanced indexation, algorithmic trading, corporate payout determination and agent-based modelling of liquidity costs, and trade strategy adaptation. The following section provides a short introduction to the individual chapters.

A. Brabazon et al. (Eds.): Natural Computing in Comput. Finance, SCI 380, pp. 1–8.
springerlink.com

1.2 The Chapters

1.2.1 Optimisation

A wide variety of NC methodologies including genetic algorithms, evolutionary strategies, differential evolution and particle swarm optimisation have been applied for optimisation purposes in finance. A particular advantage of these methodologies is that, if applied properly, they can cope with 'difficult' search spaces. Optimisation problems abound in finance and the first four chapters illustrate a range of interesting applications of natural computing algorithms for financial optimisation, specifically, portfolio selection, option model calibration, classifier construction and financial trading.

A key issue for investors wishing to trade in derivatives is the determination of the fair price for the derivative(s) of interest. For some standard derivatives closed form pricing equations have been determined (e.g., the Black - Scholes - Merton model for pricing European options). The traditional approach to pricing a derivative is [10]:

- specify a stochastic process for the underlying asset(s),
- derive the pricing equation for the derivative (using a no-arbitrage argument), and finally,
- price the derivative by solving the pricing equation.

Of course, this approach can be difficult to implement, as the relevant stochastic process may be imperfectly understood, and the pricing equation may be too difficult to solve analytically. In the latter case, there is scope to use tools such as Monte Carlo (MC) simulation to estimate the expected payoff and the associated payoff risk for the derivative. In valuing a complex derivative using MC, the typical approach is to randomly generate a set of independent price paths for each security underpinning the derivative, then compute the present value of the payoff to the derivative under each set of these price paths. The simulation process is repeated multiple times and the distribution of the payoffs is considered to characterise the derivative. A critical issue in applying a MC approach is the correct design of the theoretical pricing model.

In model calibration, the objective is to estimate the parameters of ('calibrate') a theoretical pricing model. The parameters are estimated by fitting the model to the relevant time series. Typically the pricing model will have a complex, non-linear structure with multiple parameters. Hence, global search heuristics can have utility in uncovering a high-quality set of parameters.

In Chapter 2 (*Calibrating Option Pricing Models with Heuristics* by Manfred Gilli and Enrico Schumann) the authors investigate the application of differential evolution and particle swarm optimisation, supplemented by adding a Nelder–Mead direct search, for the calibration of Heston's stochastic volatility model and Bates's model. They discuss how to price options under these models and how to calibrate the parameters of the models with heuristic techniques. In both cases, finding the values of the model parameters that make them consistent with market prices means solving a non-convex optimisation problem.

The results indicate that while good price fits could be achieved with all methods, the convergence of parameter estimates was much slower and in some cases (for the jump parameters of the Bates model) there was no convergence. This, highlights an important issue, that while different parameter values may lead to good overall fits in terms of prices, these different parameters may well imply very different Greeks, or have a more marked influence on prices of exotic options. The findings underscore the point that modellers in quantitative finance should be wary of over-reliance on model fit without adequate consideration of the stability of the estimates of the model parameters.

A common application in financial decision-making is the correct classification of some item. For example, is this customer a good credit risk? Often, even where the decision faced is an apparently simple binary one ('go' or 'no go'), and where there are relevant historical examples of previous outcomes, there will be differing numbers of training cases for each outcome. In an extreme case, there may few (or no) examples of one possible outcome.

Chapter 3 (*A Comparison Between Nature-Inspired and Machine Learning Approaches to Detecting Trend Reversals in Financial Time Series* by Antonia Azzini, Matteo De Felice and Andrea G. B. Tettamanzi) focusses on the detection of turning points, or 'trend reversal' points, in financial-time series, an issue of obvious relevance for trading strategies. Two methods from natural computing are employed, namely Particle Swarm Optimisation (PSO) and the negative selection artificial immune system algorithm. A particular feature of the natural immune system is that it uses one-class learning (learning the 'normal state') and hence can detect changes to an abnormal state even though the new state has never been seen before. One class learning can be particularly useful in real-world classification problems where limited examples of a class of interest exist. In the chapter, both natural computing methods are used in order to create a high-quality set of hyperspherical detectors, the PSO doing this using a 'positive selection' mechanism whereas the latter algorithm uses a 'negative selection' process, defining the anomalous space as the complement of the normal one. The performances of the two approaches are compared against those obtained from traditional machine learning techniques and both methods are found to give interesting results with respect to traditional techniques.

Passive index funds have captured a major share of investment funds in recent years. Although the basic idea of mimicking the performance of a market index at low cost is attractive, a number of problems emerge when trying to implement this in practice. The transaction costs involved in tracking changes in the index composition can be non-trivial and other possible constraints such as cardinality constraints, floor/ceiling mandate constraints, and lot size constraints result in a complex optimisation problem. A variant on the passive index strategy is that of *enhanced indexation* where the fund manager is allowed to deviate from a strict index tracking strategy in order to seek 'alpha'. Typically these funds will have a synthetic objective function which trades off tracking error against the possibility of enhanced returns.

In Chapter 4 (*A Soft Computing Approach to Enhanced Indexation* by Nikos S. Thomaidis) the author addresses this area, by structuring enhanced indexation strategies using non-typical objectives for investment performance. These focus, for example, on the frequency by which the enhanced portfolio delivers positive return relative to the benchmark, thus revealing alternative risk aspects of the investment strategy. The methodology adopted is *fuzzy mathematical multi-objective programming*, with the weights in the final investment portfolio being determined using three nature-inspired heuristics, namely, simulated annealing, genetic algorithms and particle swarm optimisation. The resulting portfolios are benchmarked against the American Dow Jones Industrial Average (DJIA) index and two other simpler heuristics for detecting good asset combinations: a Monte Carlo combinatorial optimisation method and an asset selection technique based on the capitalisation and the beta coefficients of index member stocks. The use of fuzzy systems concepts in the chapter is also worthy of note, as fuzzy systems allow for the incorporation of domain knowledge even when that knowledge can only be expressed in general terms (for example, the expertise is tacit) by the expert.

A practical problem that can arise in the application of evolutionary techniques to large datasets (such as in some financial modelling applications) is that computational resources can become an issue. One strand of recent work which has sought to alleviate this issue is to use multi-core graphics processors to parallelise aspects of the evolutionary process. For example, the calculation of fitness (which is usually the most time consuming step of an evolutionary algorithm) can be distributed across multiple cores and performed in parallel. Chapter 5 (*Parallel Evolutionary Algorithms for Stock Market Trading Rule Selection on Many-Core Graphics Processors* by Piotr Lipinski) illustrates this idea and uses GPU parallel processing, combined with a hybrid evolutionary algorithm, to improve the process of rule selection in stock market trading decision support systems. Experiments carried out on data from the Paris Stock Exchange illustrate the potential for the system.

1.2.2 Model Induction

While financial optimisation applications of natural computing are important and plentiful, the underlying model or data generating process is not always known in many applications. Hence, the task is often to 'recover' or discover an underlying model from a dataset. This is usually a difficult task as both the model structure and associated parameters must be uncovered. A variety of machine learning techniques have been developed over past decades which are capable of model induction, ranging from black-box techniques such as feedforward neural networks to techniques such as genetic programming which can embed existing domain knowledge and which are capable of producing human-readable output / models.

A particular challenge when seeking to model financial markets (for example, for trading purposes) is that they do not present a stationary environment. A common - but facile - approach is to ignore this, and 'train' a trading model based on data from a financial market of interest over a lengthy period. Unsurprisingly, models based on this approach rarely perform well out of sample. A more sophisticated approach is to

attempt to build a suite of models, where each model is tailored for use in a specific 'state' of the market. Chapter 6 (*Regime-Switching Recurrent Reinforcement Learning in Automated Trading* by Dietmar Maringer and Tikesh Ramtohul) introduces a powerful machine learning technique, reinforcement learning (RL), which focuses on goal-directed learning from interaction. It is a way of programming agents by reward and punishment without needing to specify how the task is to be achieved. In other words, the learning process does not require target outputs, unlike supervised learning techniques. RL can be used to find approximate solutions to stochastic dynamic programming problems and it can do so in an online fashion. In the last decade it has attracted growing interest in the computational finance community, especially for the design of trading systems. Typically, in these applications, a recurrent RL (RRL) approach is adopted as in practice investment performance depends on sequences of interdependent decisions.

In this chapter, the authors implement a variant on the RRL model - a 'regime-switching' recurrent reinforcement learning (RSRRL) model. The regime-switching component adds 'context-sensitivity' to the RRL and could therefore produce better trading strategies which are sensitive to current market conditions. The chapter therefore compares the performance of RRL and RSRRL approaches for the purposes of uncovering successful trading strategies. The results indicate that the RSRRL models yield higher Sharpe ratios than the standard RRL in-sample but struggle to reproduce the same performance levels out of sample. The authors suggest that the lack of in and out of sample correlation is due to a drastic change in market conditions (out of sample), and demonstrate that the RSRRL can consistently outperform the RRL only when certain conditions are present.

One of the most studied evolutionary methodologies is that of genetic programming (GP) [6]. GP is a population-based search algorithm which starts from a high-level statement of what is required and automatically creates a computer programme to solve the problem. GP belongs to the field of *Evolutionary Automatic Programming*. The term is used to refer to systems that adopt evolutionary computation to automatically generate computer programmes. More generally, a computer programme can be considered as a list of rules or as a model.

In Chapter 7 (*An Evolutionary Algorithmic Investigation of US Corporate Payout Policy Determination* by Alexandros Agapitos, Abhinav Goyal and Cal Muckley) GP is used to model the corporate payout policy for US listed companies. The term corporate payout policy relates to the disbursing of cash, by a corporation, to shareholders by way of cash dividends and/or share repurchases. Clearly, alongside investment and capital structure optimisation this is a major decision facing firms which has obvious implications for their financing. In the extant literature a variety of classical statistical methodologies have been adopted, foremost among these is the method of panel regression modelling. In this chapter, the authors inform their model specifications and coefficient estimates using a genetic program. The resulting model captures effects from a wide range of pertinent proxy variables related to the agency cost-based life cycle theory, the signalling theory and the catering theory of corporate payout policy determination. The findings indicate the predominant importance of the agency-cost based life cycle theory and provide important new

insights concerning the influence of firm size, the concentration of firm ownership and cash flow uncertainty with respect to corporate payout policy determination in the United States.

A perennial question when constructing a model using training data is 'how well will this model generalise out-of-sample'? Of course, there are many aspects to this issue including how representative is the training data, is the system of interest stationary, and when should I stop training in order to avoid 'overfit'? The latter is a significant problem when dealing with financial datasets which typically have a low signal-to-noise ratio (consider for example, the problem of trying to identify a 'good' trading model). Although the issue of overfitting has drawn significant attention in traditional econometrics, and in computational intelligence fields such as neural networks, there has been less study of this issue in GP. In Chapter 8 (*Tackling Overfitting in Evolutionary-driven Financial Model Induction* by Clíodhna Tuite, Alexandros Agapitos, Michael O'Neill and Anthony Brabazon) this issue is considered, with particular attention being paid to the classical method of 'early-stopping'. Using controlled experiments, it is demonstrated that this method does not always identify the correct point at which to stop training such that the generalisation performance of the evolved model is maximised. The chapter introduces a new metric for the determination of the optimal stopping point using the correlation between training and validation fitness, stopping training when this value drops below a predefined threshold,and illustrates that this can perform well.

1.2.3 Agent-Based Modelling

Simulation approaches, such as agent-based modelling, have become a significant research tool in computational finance and economics. The modelling of markets using agent-based approaches provides researchers with a powerful tool to examine the effects of issues including different market mechanisms and differing agent-learning mechanisms on the resulting market behaviour. A particular advantage of an agent-based simulation-based approach is that many sample paths through time can be generated and hence, if the models are sufficiently realistic, it may be possible to use them to test various scenarios or trading systems / execution strategies [12]. Agent-based approaches can complement analytical models as the combination of the two allows us to:

> to check the validity of assumptions needed in an analytical model. On the other hand, an analytical model can suggest reasonable alternatives to investigate in a simulation study. [8] (p. 115)

Of course, as with any simulation-based methodology, potential criticisms must be anticipated. Kleindorfer *et al.* [7] define *simulate* as *...to build a likeness ...* (p. 1087) and note that the *...question of that likeness is never far behind ....* Hence it is important to ground agent-based models in existing domain knowledge and to test them robustly.

Chapter 9 (*An Order-Driven Agent-Based Artificial Stock Market to Analyze Liquidity Costs of Market Orders in the Taiwan Stock Market* by Yi-Ping Huang, Shu-Heng Chen, Min-Chin Hung and Tina Yu) provides an excellent example of the

careful development of an agent-based model. The study develops an order-driven agent-based artificial stock market to analyse the liquidity costs of market orders in the Taiwan Stock Market (TWSE) and crucially, seeks to calibrate the model to real-world data. In the chapter the authors develop a variant on the model proposed by Daniels, Farmer, Gillemot, Iori and Smith [2] and use this to simulate the liquidity costs on the Taiwan Stock Market. The model is tested using data from 10 stocks from the market and the model-simulated liquidity costs were found to be higher than those of the TWSE data. A number of possible factors that have contributed to this result are identified.

A continuing challenge to all participants in financial markets is the choice and adaptation of suitable trading and investment strategies. Chapter 10 (*Market Microstructure: A Self-Organizing Map Approach to Investigate Behavior Dynamics under an Evolutionary Environment* by Michael Kampouridis, Shu-Heng Chen and Edward Tsang) presents a agent-based model, wherein agents are capable of adapting their trading strategies over time. The chapter investigates the behavior dynamics in financial markets in order to determine whether 'dinosaurs' (or former successful technical trading strategies) can return. The results, based on data from four financial markets, indicate that because of the changing nature of those markets, agents' trading strategies need to continuously adapt in order to remain effective and that 'dinosaurs' do not return. This supports a claim that market behaviour is non-stationary and non-cyclical. Strategies from the past cannot be successfully re-applied to future periods, unless they have co-evolved with the market.

1.3 Conclusions

In this the fourth volume of the Natural Computing and Computational Finance book series, we present the latest studies at the frontier of natural computing, agent-based modeling and finance and economics. We hope the reader will be as excited as we are with the continued development and diversity of research in this domain. In addition, we hope the contents of the chapters that follow will inspire others to try their hand at the challenging set of real world problems presented when modeling financial and economic systems using natural computing methods. In turn we look forward to the future advances made in understanding the natural world and the algorithms inspired by her.

References

1. Brabazon, A., O'Neill, M.: Biologically Inspired Algorithms for Financial Modelling. Springer, Berlin (2006)
2. Brabazon, A., O'Neill, M. (eds.): Natural Computing in Computational Finance. Springer, Berlin (2008)
3. Brabazon, A., O'Neill, M. (eds.): Natural Computing in Computational Finance, vol. 2. Springer, Berlin (2009)

4. Brabazon, A., O'Neill, M., Maringer, D. (eds.): Natural Computing in Computational Finance, vol. 3. Springer, Berlin (2010)
5. de Castro, L.N.: Fundamentals of natural computing: an overview. Physics of Life Reviews 4(1), 1–36 (2007)
6. Koza, J.R.: Genetic programming - on the programming of computers by means of natural selection. MIT Press, Cambridge (1992)
7. Kleindorfer, G., O'Neill, L., Ganeshan, R.: Validation in Simulation: Various Positions in the Philosophy of Science. Management Science 44(8), 1087–1099 (1998)
8. Law, A., Kelton, W.: Simulation Modeling & Analysis. McGraw-Hill, New York (1991)
9. Maringer, D.: Portfolio Management with Heuristic Optimization. Springer, Berlin (2005)
10. Noe, T., Wang, J.: The Self-evolving Logic of Financial Claim Prices. In: Chen, S.-H. (ed.) Genetic Algorithms and Genetic Programming in Computational Finance, pp. 249–262. Kluwer Academic Publishers, Dordrecht (2002)
11. Tesfatsion, L., Judd, K. (eds.): Handbook of Computational Economics Volume 2: Agent-Based Computational Economics. North-Holland, Amsterdam (2006)
12. Cui, W., Brabazon, A., O'Neill, M.: Evolutionary Computation and Trade Execution. In: Brabazon, A., O'Neill, M., Maringer, D.G. (eds.) Natural Computing in Computational Finance. Studies in Computational Intelligence, vol. 293, pp. 45–62. Springer, Heidelberg (2010)
13. Wong, B., Lai, V., et al.: A bibliography of neural network business applications research: 1994-1998. Computers and Operations Research 27, 1045–1076 (2000)

2

Calibrating Option Pricing Models with Heuristics

Manfred Gilli[1] and Enrico Schumann[2]

[1] University of Geneva, Switzerland
manfred.gilli@unige.ch
[2] VIP Value Investment Professionals AG, Zug, Switzerland
es@vipag.com

Summary. Calibrating option pricing models to market prices often leads to optimisation problems to which standard methods (such as those based on gradients) cannot be applied. We investigate two models: Heston's stochastic volatility model, and Bates's model which also includes jumps. We discuss how to price options under these models, and how to calibrate the parameters of the models with heuristic techniques.

2.1 Introduction

Implied volatilities obtained by inverting the Black–Scholes–Merton (BSM) model vary systematically with strike and maturity; this relationship is called the volatility surface. Different strategies are possible for incorporating this surface into a model. We can accept that volatility is not constant across strikes and maturities, and directly model the volatility surface and its evolution. With this approach we assume that a single underlier has different volatilities which is internally not consistent; but still, it is the approach that is mostly used in practice. An alternative is to model the option prices such that the BSM-volatility surface is obtained, for instance by including locally-varying volatility [13, 16], jumps, or by making volatility stochastic. In this chapter, we look into models that follow the latter two approaches, namely the models of [4] and [26].

As so often in finance, the success of the BSM model stems not so much from its empirical quality, but from its computational convenience. This convenience comes in two flavours. Firstly, we have closed-form pricing equations (the Gaussian distribution function is not available analytically, but fast and precise approximations exist). Secondly, calibrating the model requires only one parameter to be determined, the volatility, which can be readily computed from market prices with Newton's method or another zero-finding technique. For the Heston and the Bates model, both tasks become more difficult. Pricing requires numerical integration, and calibration requires the modeller to find five and eight parameters instead of only one for BSM.

In this chapter, we will look into the calibration of these models. Finding parameters that make the models consistent with market prices means solving a non-convex optimisation problem. We suggest the use of optimisation heuristics and more specifically we show that Differential Evolution and Particle Swarm Optimisation are both able to give good solutions

A. Brabazon et al. (Eds.): Natural Computing in Comput. Finance, SCI 380, pp. 9–37.
springerlink.com

to the problem. The chapter is structured as follows. In Section 2.2 we discuss how to price options under the Heston and the Bates model. Fast pricing routines are important since the suggested heuristics are computationally intensive; hence to obtain calibration results in a reasonable period of time, we need to be able to evaluate the objective function (which requires pricing) speedily. Section 2.3 details how to implement the heuristics for a calibration problem, Section 2.4 discusses several computational experiments and their results and the chapter is concluded in Section 2.5.

2.2 Pricing with the Characteristic Function

There are several generic approaches to price options. The essence of BSM is a no-arbitrage argument; it leads to a partial differential equation that can be solved numerically or, in particular cases, even analytically. A more recent approach builds on the characteristic function of the (log) stock price. European options can be priced using the following equation [2, 34]:[1]

$$C_0 = \mathrm{e}^{-q\tau} S_0 \Pi_1 - \mathrm{e}^{-r\tau} X \Pi_2 \tag{2.1}$$

where C_0 is the call price today (time 0), S_0 is the spot price of the underlier, and X is the strike price; r and q are the riskfree rate and dividend yield; time to expiration is denoted τ. The Π_j are calculated as

$$\Pi_1 = \frac{1}{2} + \frac{1}{\pi} \int_0^\infty \mathsf{Re}\left(\frac{\mathrm{e}^{-\mathrm{i}\omega \log(X)} \phi(\omega - \mathrm{i})}{\mathrm{i}\omega \phi(-\mathrm{i})} \right) \mathrm{d}\omega \,, \tag{2.2a}$$

$$\Pi_2 = \frac{1}{2} + \frac{1}{\pi} \int_0^\infty \mathsf{Re}\left(\frac{\mathrm{e}^{-\mathrm{i}\omega \log(X)} \phi(\omega)}{\mathrm{i}\omega} \right) \mathrm{d}\omega \,. \tag{2.2b}$$

We define $\Pi_j^* \equiv \pi(\Pi_j - 1/2)$ for the integrals in these equations. The symbol ϕ stands for the characteristic function of the log stock price; the function $\mathsf{Re}(\cdot)$ returns the real part of a complex number. For a given ϕ we can compute Π_1 and Π_2 by numerical integration, and hence obtain option prices from Equation (2.1).

2.2.1 Black–Scholes–Merton

In a BSM world, the stock price S_t under the risk-neutral measure follows

$$\mathrm{d}S_t = (r - q) S_t \mathrm{d}t + \sqrt{v} S_t \mathrm{d}z_t \tag{2.3}$$

where z is a Wiener process [5]. The volatility $\sqrt{v}$ is constant. The well-known pricing formula for the BSM call is given by

$$C_0 = \mathrm{e}^{-q\tau} S_0 \mathrm{N}(d_1) - X \mathrm{e}^{-r\tau} \mathrm{N}(d_2) \tag{2.4}$$

[1] Variants on this pricing formula exist, i.e., different approaches to price with the characteristic function. See [6], [7],[15], [18] and [29].

with

$$d_1 = \frac{1}{\sqrt{v\tau}}\left(\log\left(\frac{S_0}{X}\right) + \left(r - q + \frac{v}{2}\right)\tau\right) \tag{2.5a}$$

$$d_2 = \frac{1}{\sqrt{v\tau}}\left(\log\left(\frac{S_0}{X}\right) + \left(r - q - \frac{v}{2}\right)\tau\right) = d_1 - \sqrt{v\tau} \tag{2.5b}$$

and $\mathrm{N}(\cdot)$ the Gaussian distribution function.

Given the dynamics of S, the log price $s_\tau = \log(S_\tau)$ follows a Gaussian distribution with $s_\tau \sim \mathcal{N}\left(s_0 + \tau(r - q - \frac{1}{2}v), \tau v\right)$, where s_0 is the natural logarithm of the current spot price. The characteristic function of s_τ is given by

$$\begin{aligned}\phi_{\mathrm{BSM}}(\omega) &= \mathsf{E}(\mathrm{e}^{\mathrm{i}\omega s_\tau}) \\ &= \exp\left(\mathrm{i}\omega s_0 + \mathrm{i}\omega\tau(r - q - \frac{1}{2}v) + \frac{1}{2}\mathrm{i}^2\omega^2 v\tau\right) \\ &= \exp\left(\mathrm{i}\omega s_0 + \mathrm{i}\omega\tau(r - q) - \frac{1}{2}(\mathrm{i}\omega + \omega^2)\tau v\right).\end{aligned} \tag{2.6}$$

Inserting (2.6) into Equation (2.1) should, up to numerical precision, give the same result as Equation (2.4).

2.2.2 Merton's Jump–Diffusion Model

Merton (1976) [30] suggested the modelling of the underlier's movements as a diffusion with occasional jumps; thus we have

$$\mathrm{d}S_t = (r - q - \lambda\mu_J)S_t\mathrm{d}t + \sqrt{v}S_t\mathrm{d}z_t + J_t S_t \mathrm{d}N_t\,. \tag{2.7}$$

N_t is a poisson counting process, with intensity λ; the J_t is the random jump size (given that a jump occurred). In Merton's model the log-jumps are distributed as

$$\log(1 + J_t) \sim \mathcal{N}\left(\log(1 + \mu_J) - \frac{\sigma_J^2}{2}, \sigma_J^2\right).$$

The pricing formula is the following [30] (p. 135):

$$C_0 = \sum_{n=0}^{\infty} \frac{e^{-\lambda'\tau}(\lambda'\tau)^n}{n!} C_0'(r_n, \sqrt{v_n}) \tag{2.8}$$

where $\lambda' = \lambda(1 + \mu_J)$ and C_0' is the BSM formula (2.4), but the prime indicates that C_0' is evaluated at adjusted values of r and v:

$$\begin{aligned} v_n &= v + \frac{n\sigma_J^2}{\tau} \\ r_n &= r - \lambda\mu_J + \frac{n\log(1 + \mu_J)}{\tau} \end{aligned}$$

The factorial in Equation (2.8) may easily lead to an overflow (`Inf`), but it is benign for two reasons. Firstly, we do not need large numbers for n, a value of about 20 is well-sufficient. Secondly (if we insist on large n), software packages like `Matlab` or `R` will evaluate `1/Inf` as zero, hence the summing will add zeros for large n. Numerical analysts prefer to replace

$n!$ by $\exp(\sum_{i=1}^{n} \log i)$ since this leads to better accuracy for large n. Again, for Merton's model this is not needed. Depending on the implementation, working with large values for n may still lead to a warning or an error, and so interrupt a computation. In R for instance, the handling of such a warning will depend on the options setting:

```
1 > options()$warn
2 [1] 0
```

This is the standard setting. Computing the factorial for a large number will result in a warning; the computation continues.

```
1 > factorial(200)
2 [1] Inf
3 Warning message:
4 In factorial(200) : value out of range in 'gammafn'
```

But with `warn` set to two, any warning will be transformed into an error. Thus:

```
1 > options(warn=2)
2 > factorial(200)
3 Error in factorial(200) :
4  (converted from warning) value out of range in 'gammafn'
```

and our computation breaks. We may want to safeguard against such possible errors: we can for instance replace the function call `factorial(n)` by its actual calculation which produces:

```
1 > options(warn=2)
2 > exp( sum(log(1:200)) )
3 [1] Inf
4 > prod(1:200)
5 [1] Inf
```

Or even simpler, as in `Matlab's` implementation of `factorial`, we can check the given value of n; if it is too large, we have it replaced by a more reasonable value.

The characteristic function of Merton's model is given by

$$\phi_{\text{Merton}} = \mathrm{e}^{A+B} \tag{2.9}$$

where

$$\begin{aligned} A &= \mathrm{i}\omega s_0 + \mathrm{i}\omega\tau(r - q - \frac{1}{2}v - \mu_J) + \frac{1}{2}\mathrm{i}^2\omega^2 v\tau \\ B &= \lambda\tau\left(\exp\left(\mathrm{i}\omega\log(1+\mu_J) - \frac{1}{2}\mathrm{i}\omega\sigma_J^2 - \omega^2\sigma_J^2\right) - 1\right), \end{aligned}$$

see [19] (chp. 5). The A-term in ϕ_{Merton} corresponds to the BSM dynamics with a drift adjustment to account for the jumps; the B-term adds the jump component. Like in the BSM case, we can compare the results from Equation (2.1) with those obtained from Equation (2.8).

2.2.3 The Heston Model

Under the Heston model [26] the stock price S and its variance v are described by

$$\mathrm{d}S_t = rS_t\mathrm{d}t + \sqrt{v_t}S_t\mathrm{d}z_t^{(1)} \tag{2.10a}$$

$$\mathrm{d}v_t = \kappa(\theta - v_t)\mathrm{d}t + \sigma\sqrt{v_t}\mathrm{d}z_t^{(2)}\,. \tag{2.10b}$$

The long-run variance is denoted θ, mean reversion speed is κ and σ is the volatility-of-volatility. The Wiener processes $z^{(\cdot)}$ have correlation ρ. For $\sigma \to 0$, the Heston dynamics approach those of BSM. A thorough discussion of the model can be found in [19]. The characteristic function of the log-price in the Heston model looks as follows, see [1]:

$$\phi_{\text{Heston}} = \mathrm{e}^{A+B+C} \tag{2.11}$$

where

$$A = \mathrm{i}\omega s_0 + \mathrm{i}\omega(r-q)\tau$$

$$B = \frac{\theta\kappa}{\sigma^2}\left((\kappa - \rho\sigma\mathrm{i}\omega - d)\tau - 2\log\left(\frac{1-g\mathrm{e}^{-d\tau}}{1-g}\right)\right)$$

$$C = \frac{\frac{v_0}{\sigma^2}\left(\kappa - \rho\sigma\mathrm{i}\omega - d\right)\left(1-\mathrm{e}^{-d\tau}\right)}{1-g\mathrm{e}^{-d\tau}}$$

$$d = \sqrt{(\rho\sigma\mathrm{i}\omega - \kappa)^2 + \sigma^2(\mathrm{i}\omega + \omega^2)}$$

$$g = \frac{\kappa - \rho\sigma\mathrm{i}\omega - d}{\kappa - \rho\sigma\mathrm{i}\omega + d}$$

With only five parameters (under the risk-neutral probability), the Heston model is capable of producing a volatility smile, see Figure 2.1.

2.2.4 The Bates Model

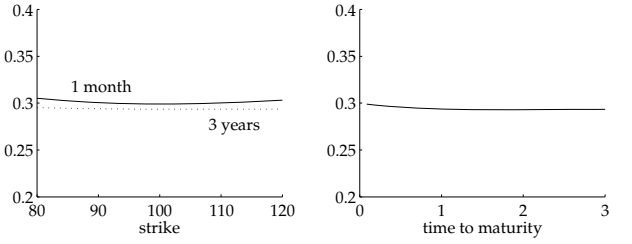
This model, described in [4], adds jumps to the dynamics of the Heston model. The stock price S and its variance v are described by

$$\mathrm{d}S_t = (r - q - \lambda\mu_J)S_t\mathrm{d}t + \sqrt{v_t}S_t\mathrm{d}z_t^{(1)} + J_tS_t\mathrm{d}N_t$$

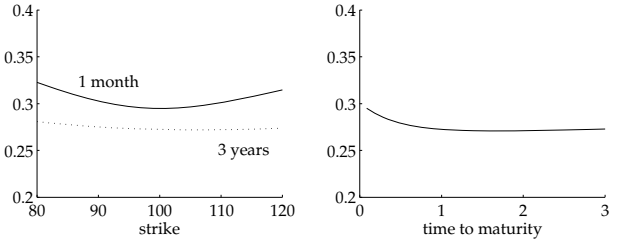
$$\mathrm{d}v_t = \kappa(\theta - v_t)\mathrm{d}t + \sigma\sqrt{v_t}\mathrm{d}z^{(2)}\,.$$

N_t is a poisson count process with intensity λ, hence the probability to have a jump of size one is $\lambda\,\mathrm{d}t$. Like Merton's model, the logarithm of the jump size J_t is distributed as a Gaussian, i.e.,

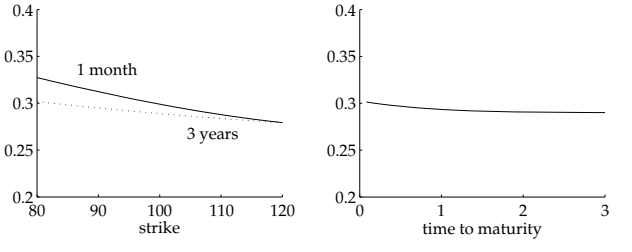
$$\log(1+J_t) = \mathcal{N}\left(\log(1+\mu_J) - \frac{\sigma_J^2}{2}, \sigma_J^2\right).$$



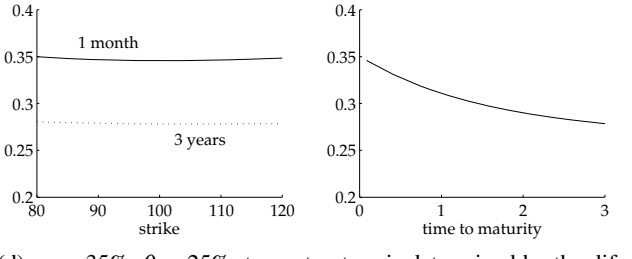
(a) The base case: $S = 100$, $r = 2\%$, $q = 2\%$, $\sqrt{v_0} = 30\%$, $\sqrt{\theta} = 30\%$, $\rho = 0$, $\kappa = 1$, $\sigma = 30\%$



(b) $\sigma = 70\%$: short-term smile (the position of the kink is controlled by ρ); often we need substantial volatility-of-volatility



(c) $\rho = -0.5$: skew (a positive correlation induces positive slope)



(d) $v_0 = 35\%$, $\theta = 25\%$: term structure is determined by the difference between current and long-run variance, and κ

Fig. 2.1 Heston model: re-creating the implied volatility surface. The graphics show the BSM implied volatilities obtained from prices under the Heston model. The panels on the right show the implied volatility of ATM options.

The characteristic function becomes [35]:

$$\phi_{\text{Bates}} = \mathrm{e}^{A+B+C+D} \tag{2.12}$$

with

$$\begin{aligned}
A &= \mathrm{i}\omega s_0 + \mathrm{i}\omega(r-q)\tau \\
B &= \frac{\theta\kappa}{\sigma^2}\left((\kappa - \rho\sigma\mathrm{i}\omega - d)\tau - 2\log\left(\frac{1-g\mathrm{e}^{-d\tau}}{1-g}\right)\right) \\
C &= \frac{\frac{v_0}{\sigma^2}\left(\kappa - \rho\sigma\mathrm{i}\omega - d\right)\left(1-\mathrm{e}^{-d\tau}\right)}{1-g\mathrm{e}^{-d\tau}} \\
D &= -\lambda\mu_J\mathrm{i}\omega\tau + \lambda\tau\left((1+\mu_J)^{\mathrm{i}\omega}\mathrm{e}^{\frac{1}{2}\sigma_J^2\mathrm{i}\omega(\mathrm{i}\omega-1)} - 1\right)
\end{aligned}$$

$$\begin{aligned}
d &= \sqrt{(\rho\sigma\mathrm{i}\omega - \kappa)^2 + \sigma^2(\mathrm{i}\omega + \omega^2)} \\
g &= \frac{\kappa - \rho\sigma\mathrm{i}\omega - d}{\kappa - \rho\sigma\mathrm{i}\omega + d}
\end{aligned}$$

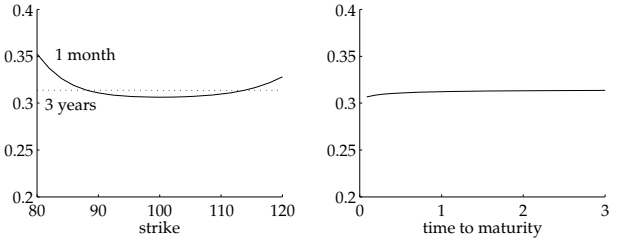
Since the jumps are assumed independent, the characteristic function is the product of ϕ_{Heston} with the function for the jump part (D). Figure 2.2 shows that adding jumps makes it easier to introduce curvature into the volatility surface, at least for short maturities.

2.2.5 Integration Schemes

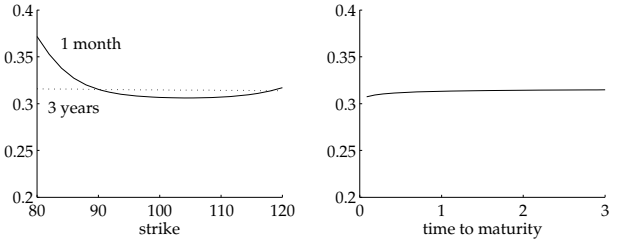
Matlab programs for pricing under the different models can be downloaded from `http://comisef.eu`. The programs use `Matlab`'s `quad` function, an adaptive algorithm based on Simpson's rule; it is reliable but slow. The pricing can be accelerated by precomputing a fixed number of nodes and weights under the given quadrature scheme. We use a Gauss–Legendre rule, see [12] and [39]; a technical report, also downloadable from `http://comisef.eu`, contains details and `Matlab` code. We experimented with alternatives like Gauss–Lobatto as well, but no integration scheme was clearly dominant over another, given the required precision of our problem (there is no need to compute option prices to eight decimals). Thus, in what follows, we do not use `quad`, but compute nodes and weights, and directly evaluate the integrals in Equations (2.2).

To test our pricing algorithms, we first investigate the BSM model and Merton's jump–diffusion model. For these models, we can compare the solutions obtained from the classical formulæ with those from integration. Furthermore, we can investigate several polar cases: for Heston with zero volatility-of-volatility we should get BSM prices; for Bates with zero volatility-of-volatility we should obtain prices like under Merton's jump diffusion (and of course Bates with zero volatility-of-volatility and no jumps should again give BSM prices).

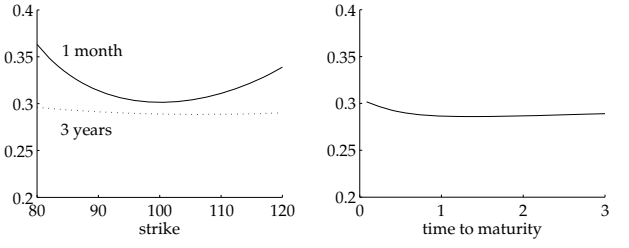
As an example of these tests, we compare here results from Equation (2.4) with results from Equation (2.1). The integrands in Equations (2.2) are well-behaved for BSM, see Figure 2.3 in which we plot $\Pi_j^* = \pi(\Pi_j - 1/2)$. The functions start to oscillate for options that are far away from the money, increasingly so for low volatility and short time to maturity. This is shown in the left panel of Figure 2.4, where we plot Π_1^* for varying strikes X and ω values (Π_2^* looks similar). The right panel of Figure 2.4 gives the same plot, but shows only the strikes from 80 to 120; here little oscillation is apparent.



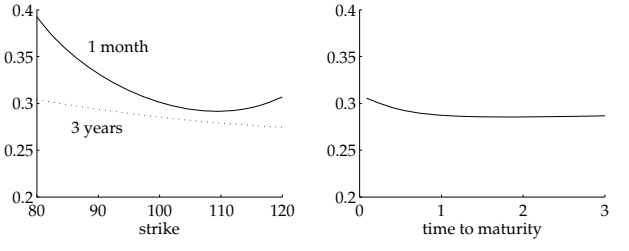
(a) The base case: $S = 100$, $r = 2\%$, $q = 2\%$, $\sqrt{v_0} = 30\%$, $\sqrt{\theta} = 30\%$, $\rho = 0$, $\kappa = 1$, $\sigma = 0.0\%$, $\lambda = 0.1$, $\mu_J = 0$, $\sigma_J = 30\%$. Volatility-of-volatility is zero, as is the jump mean



(b) $\mu_J = -10\%$: more asymmetry



(c) $\theta = 70\%$: stochastic volatility included



(d) $\mu_J = -10\%$, $\theta = 70\%$, $\rho = -0.3$

Fig. 2.2 Bates model: re-creating the implied volatility surface. The graphics show the BSM implied volatilities obtained from prices under the Bates model. The panels on the right show the implied volatility of ATM options.

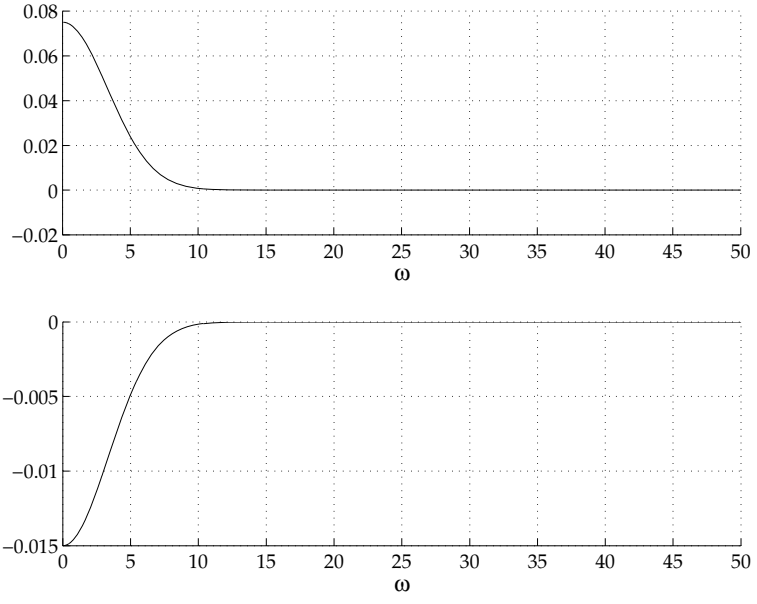


Fig. 2.3 Π_1^* and Π_2^* for BSM ($S = 100,\ X = 100,\ \tau = 1,\ \sqrt{v} = 0.3,\ r = 0.03$)

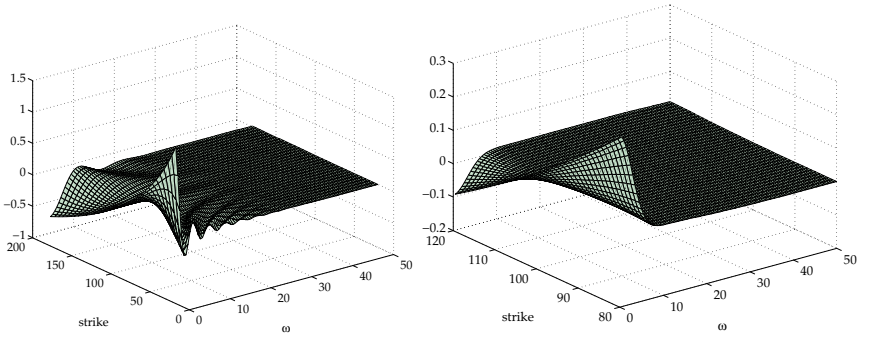


Fig. 2.4 Π_1^* for BSM with varying strikes ($S = 100,\ \tau = 1/12,\ \sqrt{v} = 0.3,\ r = 0.03$)

Gauss–Legendre quadrature is applicable to finite intervals, but the integrals in Equations (2.2) decay so rapidly to zero that we can also evaluate them up to a cutoff point, which we set to 200. Figure 2.5 shows the relative pricing errors for the BSM case with 20 nodes (left) and 100 nodes (right). Note that here we are already pricing a whole matrix of options (different strikes, different maturities). This matrix is taken from the experiments described in the next section. Already with 100 nodes the pricing errors are in the range of 10^{-13}, i.e., practically zero.

The behaviour of the integrals is similar for other characteristic functions. For the Heston model, examples for Π_j^* are given in Figure 2.6. If we let the volatility-of-volatility go to

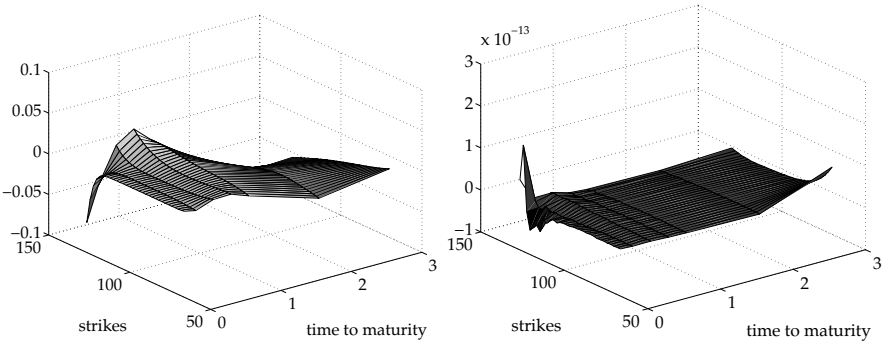


Fig. 2.5 Relative pricing errors compared with analytical BSM: a Gauss-rule with 20 nodes (left) and 100 nodes (right)

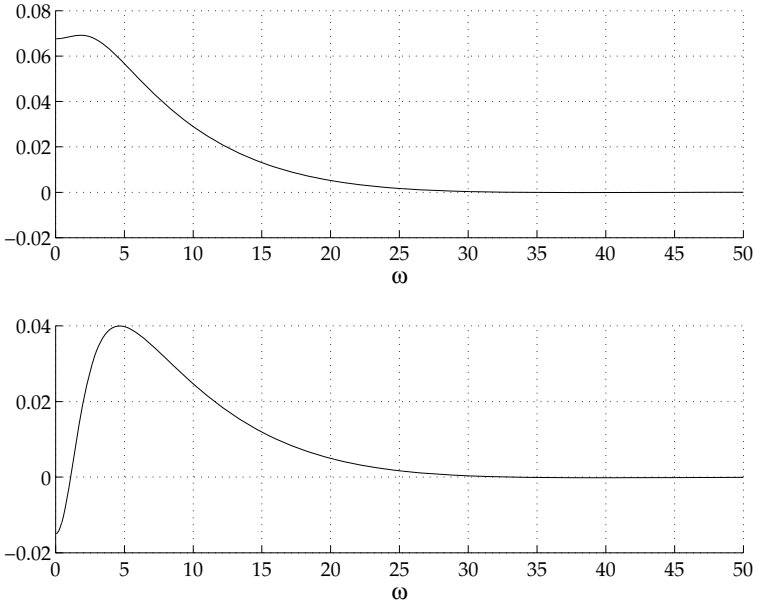


Fig. 2.6 Π_1^* and Π_2^* for Heston model ($S = 100,\ X = 100,\ \tau = 1,\ \sqrt{v_T} = 0.3, \sqrt{v_0} = 0.3,\ r = 0.03,\ \kappa = 0.2,\ \sigma = 0.8,\ \rho = -0.5$)

zero, the functions exactly resemble those of the corresponding BSM case. Figure 2.7 shows Π_1^* for different strikes, analogously to Figure 2.4.

Two more remarks: firstly, integration rules like Gauss–Legendre (or others, e.g., Clenshaw–Curtis) prescribe to sample the integrand at points that cluster around the endpoints of the interval. This happens because essentially a Gauss rule approximates the function to be integrated by a polynomial, and then integrates this polynomial exactly. Gauss rules are even optimal in the sense that for a given number of nodes, they integrate exactly a polynomial of

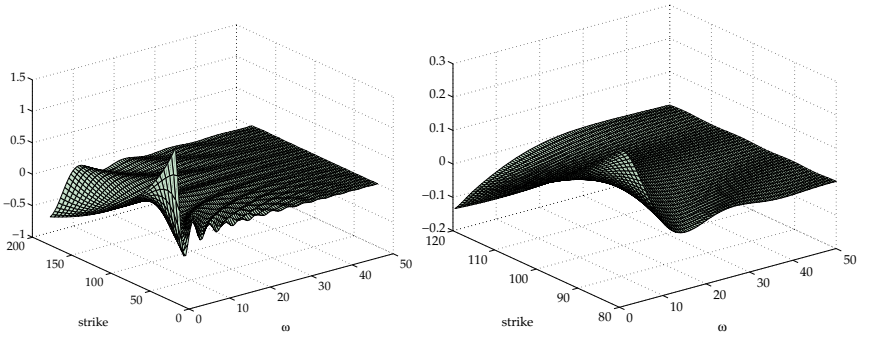


Fig. 2.7 Π_1^* for Heston model with varying strikes ($S = 100,\ \tau = 1/12,\ \sqrt{v_T} = 0.3, \sqrt{v_0} = 0.3,\ r = 0.03,\ \kappa = 0.2,\ \sigma = 0.8,\ \rho = -0.5$)

highest order possible. For an oscillating function, however, we may need a very-high–order polynomial to obtain a good approximation, hence alternative rules may be more efficient for such functions [24]. In our experiments this oscillation never caused problems. Furthermore, in Figure 2.4 and Figure 2.7 the strikes we computed range from 20 to 180 (with spot at 100). The potentially difficult cases are options with delta zero or delta one, which are not the instruments that are difficult to price.

Secondly, let us stress why (or rather, when) the efficiency of the quadrature scheme is important. The calibration algorithms that we describe in the next section are iterative methods that move through the space of possible parameters; in every iteration, a given parameter set is used to compute theoretical option prices which are then compared with observed prices. Since we will run through thousands of iterations, any speedup in our integration scheme and hence in pricing the options is greatly magnified. True, for practical applications efficiency in the sense of fast programs is eventually a constraint, not a goal per se. If we need to compute a solution to some problem within 5 minutes, and we have a program that achieves this, there is no need to spend days (or more, probably) to make the program run in 30 seconds. But even outside high-frequency trading environments, there are financial applications for which a lack of speed becomes a hindrance – just think of testing (or 'backtesting'). We will rarely devise a model, implement it and start to use it; we will start by testing a model on historic data. The faster our programs are, the more we can test. Also, we may want to scale up the model later on, maybe with more data or more assets, hence faster routines become desirable.

2.3 Calibrating Model Parameters

Calibrating an option pricing model means to find parameters such that the model's prices are consistent with market prices, leading to an optimisation problem of the form

$$\min \sum_{i=1}^{M} \frac{\left|C_i^{\text{model}} - C_i^{\text{market}}\right|}{C_i^{\text{market}}} \tag{2.13}$$

where M is the number of market prices. Alternatively, we could specify absolute deviations, use squares instead of absolute values, or introduce weighting schemes. The choice of the objective function depends on the application at hand; ultimately, it is an empirical question

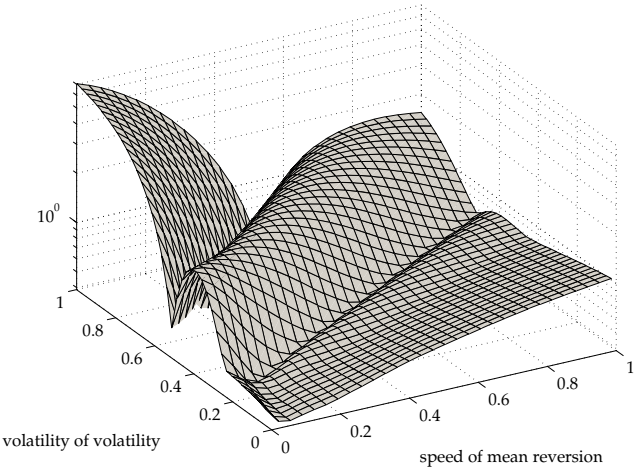


Fig. 2.8 A search space for the Heston model

to determine a good objective function. Since here we are interested in numerical aspects, we will use specification (2.13). Figure 2.8 shows an example objective function for the Heston model (on a log scale) when varying two parameters – mean reversion κ and volatility-of-volatility σ – while holding the others fixed. The problem is not convex, and standard methods (e.g., based on derivatives of the objective function) may fail. The optimisation problem can be tackled in different ways, of instance by local smoothing or regularisation of the objective function (see for instance [8, 9] or [10]). We deploy heuristic methods, Differential Evolution [37] and Particle Swarm Optimisation [17], to solve problem (2.13). Heuristic methods have the advantage that we can clearly differentiate between the financial and empirical aspects of the model on the one hand, and its numerical solution on the other. For instance, we may well decide that smoothing the objective function is a good idea if our data is subject to sampling error. In other words, smoothing would be employed for empirical reasons; not because we are forced to do so because our numerical methods are not capable of handling the problem. For a numerical analyst, an optimisation model is well-posed if it has a solution, the solution is unique, and the model is not too sensitive, i.e., small perturbations in the data do not change the solution disproportionately. Using regularisation we can transform ill-posed into well-posed models, but of course that means that we change the model that we solve. Yet if we are interested in the financial application, it matters for what purpose we have set up the model in the first place. For instance, if we just aim to interpolate option prices, the uniqueness of a solution may not be our primary concern, only the model fit. See [23] for a further discussion.

When we evaluate (2.13), we price not just one option, but a whole array of different strikes and different maturities. But for a given set of parameters that describe the underlying process of the model, the characteristic function ϕ only depends on the time to maturity, not on the strike price. This suggests that speed improvements can be achieved by preprocessing those terms of ϕ that are constant for a given maturity, and then compute the prices for all strikes for this maturity, see [28] for a discussion, see Algorithm 2.1 for a summary.

Algorithm 2.1 Computing the Prices for a Given Surface

set parameters, set $\mathscr{T}$ = maturities, set $\mathscr{X}$ = strikes
for $\tau \in \mathscr{T}$ **do**
  compute characteristic function ϕ
  for $X \in \mathscr{X}$ **do**
    compute price for strike X, maturity τ
  end for
end for
compute objective function

2.3.1 Differential Evolution

Differential Evolution (DE) is described in detail in [37]. DE evolves a population of n_P solutions, stored in real-valued vectors of length p (p is five for Heston, eight for Bates). In every generation k, the algorithm creates n_P new candidate solutions; one new solution for each existing one. Such candidate solutions are constructed by taking the difference between two other solutions, weighting this difference, and adding the weighted difference to a third solution. Then an element-wise crossover takes place between the auxiliary solutions $P^{(v)}$ and the original solutions. If such a final candidate solution is better than the original solution, it replaces it; if not, the old solution is kept.

Algorithm 2.2 summarises the technique. (Notation: n_G - number of generations; F - weight parameter; CR - crossover probability; P - population (a matrix of size $p \times n_P$); $\ell_{(\cdot)}$ - integers; F - objective function; ζ - a random variate with uniform distribution on $[0\ 1]$).

Algorithm 2.2 Differential Evolution

set parameters n_P, n_G, F and CR
initialise population $P^{(1)}_{j,i}$, $j = 1,\ldots,p$, $i = 1,\ldots,n_P$
for $k = 1$ to n_G **do**
  $P^{(0)} = P^{(1)}$
  for $i = 1$ to n_P **do**
    generate $\ell_1, \ell_2, \ell_3 \in \{1,\ldots,n_P\}$, $\ell_1 \neq \ell_2 \neq \ell_3 \neq i$
    compute $P^{(v)}_{\cdot,i} = P^{(0)}_{\cdot,\ell_1} + \mathtt{F} \times (P^{(0)}_{\cdot,\ell_2} - P^{(0)}_{\cdot,\ell_3})$
    for $j = 1$ to p **do**
      if $\zeta < \mathtt{CR}$ **then** $P^{(u)}_{j,i} = P^{(v)}_{j,i}$ **else** $P^{(u)}_{j,i} = P^{(0)}_{j,i}$
    end for
    if $F(P^{(u)}_{\cdot,i}) < F(P^{(0)}_{\cdot,i})$ **then** $P^{(1)}_{\cdot,i} = P^{(u)}_{\cdot,i}$ **else** $P^{(1)}_{\cdot,i} = P^{(0)}_{\cdot,i}$
  end for
end for
find best solution $gbest = \text{argmin}_i\ F(P^{(1)}_{\cdot,i})$
solution $= P^{(1)}_{\cdot,gbest}$

2.3.2 Particle Swarm Optimisation

In Particle Swarm Optimisation (PS; [17]), we again have a population that comprises n_P solutions, stored in real-valued vectors. In every generation, a solution is updated by adding another vector called velocity v_i. We may think of a solution as a position in the search space, and of velocity as a direction into which the solution is moved. Velocity changes over the course of the optimisation, the magnitude of change is the sum of two components: the direction towards the best solution found so far by the particular solution, $Pbest_i$, and the direction towards the best solution of the whole population, $Pbest_{\text{gbest}}$. These two directions are perturbed via multiplication with a uniform random variable ζ and constants $c_{(\cdot)}$, and summed, see Statement 7. The vector so obtained is added to the previous v_i, the resulting updated velocity is added to the respective solution. In some implementations, the velocities are reduced in every generation by setting the parameter δ, called inertia, to a value smaller than unity.

Algorithm 2.3 details the procedure. (Notation: n_G - number of generations; P - population (a matrix of size $p \times n_P$); F - objective function; F_i - objective function value associated with the ith solution; ζ - a random variate with uniform distribution on $[0\ 1]$).

Algorithm 2.3 Particle Swarm

1: set parameters n_P, n_G, δ, c_1 and c_2
2: initialise particles $P_i^{(0)}$ and velocity $v_i^{(0)}$, $i = 1,\ldots,n_P$
3: evaluate objective function $F_i = F(P_i^{(0)})$, $i = 1,\ldots,n_P$
4: $Pbest = P^{(0)}$, $Fbest = F$, $Gbest = \min_i(F_i)$, $gbest = \text{argmin}_i(F_i)$
5: **for** $k = 1$ to n_G **do**
6: **for** $i = 1$ to n_P **do**
7: $\triangle v_i = c_1 \times \zeta_1 \times (Pbest_i - P_i^{(k-1)}) + c_2 \times \zeta_2 \times (Pbest_{\text{gbest}} - P_i^{(k-1)})$
8: $v_i^{(k)} = \delta v^{(k-1)} + \triangle v_i$
9: $P_i^{(k)} = P_i^{(k-1)} + v_i^{(k)}$
10: **end for**
11: evaluate objective function $F_i = F(P_i^{(k)})$, $i = 1,\ldots,n_P$
12: **for** $i = 1$ to n_P **do**
13: **if** $F_i < Fbest_i$ **then** $Pbest_i = P_i^{(k)}$ and $Fbest_i = F_i$
14: **if** $F_i < Gbest$ **then** $Gbest = F_i$ and $gbest = i$
15: **end for**
16: **end for**
17: solution $= P_{\cdot,gbest}^{(n_G)}$

2.3.3 A Simple Hybrid

Population-based methods like PS and DE are often effective in exploration: they can quickly identify promising areas of the search space; but then these methods converge only slowly. In the literature we thus often find combinations of population-based search with local search

(in the sense of a trajectory method that evolves only a single solution); an example are Memetic Algorithms [31]. We also test a simple hybrid based on this idea; it combines DE and PS with a direct search component. In the classification systems of [38] or [40], this is a high-level relay hybrid.

Preliminary tests suggested that the objective function is often flat, thus different parameter values give similar objective function values. This indicates that (i) our problem may be sensitive to small changes in the data when we are interested in precise parameter estimates; and that (ii) if we insist on computing parameters precisely, we may need either many iterations, or an algorithm with a large step size. Thus, as a local search strategy, we use the direct search method of [32] as implemented in `Matlab's fminsearch`. This algorithm can change its step size; it is also robust in case of noisy objective functions (e.g., functions evaluated by numerical techniques that may introduce truncation error, as could be the case here). The hybrid is summarised in Algorithm 2.4.

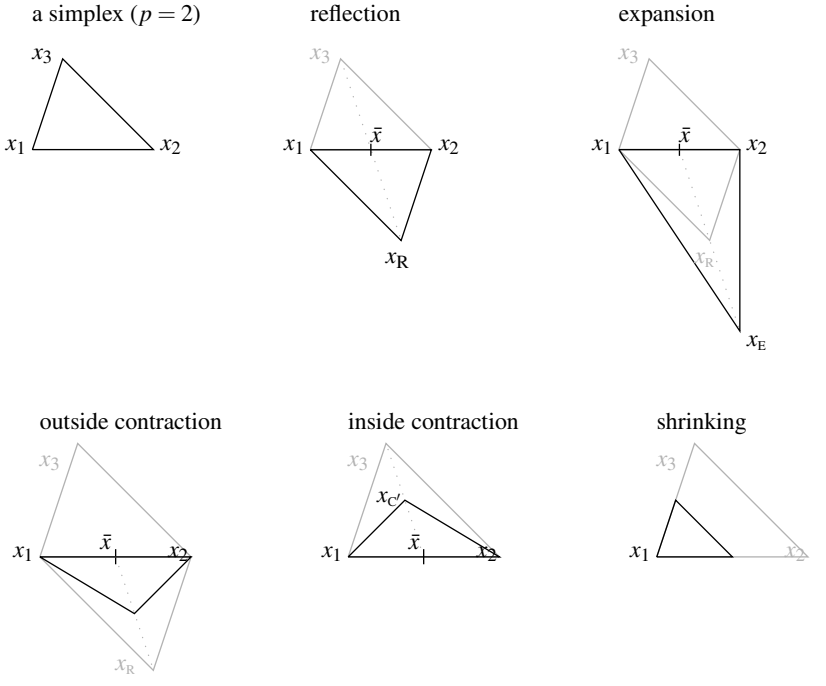
Algorithm 2.4 Hybrid Search

```
set parameters for population-based method
for k = 1 to n_G do
    do population-based search
    if local search then
        select n_S solutions as starting values for local search
        for each selected solution do
            perform local search
            update solution in population
        end for
    end if
end for
```

For an implementation, we need to decide how often we start a local search, how many solutions we select, and how we select them. In the extreme, with just one generation and $n_S = n_P$, we would have a simple restart strategy for the local search method.

Nelder–Mead Search

Spendley *et al.* (1962) [36] suggested to code a solution x as a simplex. A simplex of dimension p consists of $p+1$ vertices (points), hence for $p = 1$ we have a line segment; $p = 2$ is a triangle, $p = 3$ is a tetrahedron, and so on. In the algorithm of [36], this simplex could be reflected across an edge, or it could shrink. Thus, the size of the simplex could change, but never its form. Nelder and Mead (1965) [32] added two more operations: now the simplex could also expand and contract; hence the simplex could change its size and its form. The possible operations are illustrated in the following figure:



Algorithm 2.5 outlines the Nelder–Mead algorithm. The notation follows [41]; when solutions are ordered as

$$x_1,\ x_2,\ \ldots, x_{p+1}$$

this means we have

$$F(x_1) < F(x_2) < \ldots < F(x_{p+1})\,.$$

We denote the objective values associated with particular solutions as $F_1\,, F_2\,, \ldots, F_{p+1}\,.$ Typical values for the parameters in Algorithm 2.5 are

$$\rho = 1, \chi = 2,\ \gamma = 1/2, \text{ and } \varsigma = 1/2;$$

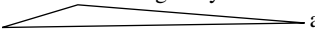
these are also used in `Matlab's fminsearch`. Matlab transforms our initial guess x into

$$\begin{array}{ccccccc}
x^{(1)} & x^{(1)}+\varepsilon x^{(1)} & x^{(1)} & x^{(1)} & \ldots & x^{(1)} \\
x^{(2)} & x^{(2)} & x^{(2)}+\varepsilon x^{(2)} & x^{(2)} & \ldots & x^{(2)} \\
x^{(3)} & x^{(3)} & x^{(3)} & x^{(1)}+\varepsilon x^{(3)} & \ldots & x^{(3)} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
x^{(p)} & x^{(p)} & x^{(p)} & x^{(p)} & \ldots & x^{(p)}+\varepsilon x^{(p)}
\end{array} \tag{2.14}$$

where the superscript $^{(i)}$ denotes the ith element of x. In the implementation used here (Matlab 2008a), ε is 0.05. If $x^{(i)}$ is zero, then $\varepsilon x^{(i)}$ is set to 0.00025.

Algorithm 2.5 Nelder–Mead Search

set ρ, χ, γ, ς
while stopping criteria not met **do**
  order points $x_1,\ldots,x_{p+1}$, compute $\bar{x} = {}^1\!/_p \sum_{i=1}^{p} x_i$
  set *shrink* = false
  $x_R = \bar{x} + \rho(\bar{x} - x_{p+1})$, $F_R = F(x_R)$ # reflect
  if $F_1 \leq F_R < F_p$ **then**
    $x^* = x_R$
  else if $F_R < F_1$ **then**
    $x_E = \bar{x} + \chi(x_R - \bar{x})$, $F_E = F(x_E)$ # expand
    if $F_E < F_R$ **then** $x^* = x_E$ **else** $x^* = x_R$
  else if $F_R \geq F_p$ **then**
    if $F_p \leq F_R < F_{p+1}$ **then**
      $x_C = \bar{x} + \gamma(x_R - \bar{x})$, $F_C = F(x_C)$ # outside contract
      if $F_C \leq F_R$ **then** $x^* = x_C$ **else** *shrink* = true
    else
      $x_{C'} = \bar{x} - \gamma(\bar{x} - x_{p+1})$, $f_{C'} = f(x_{C'})$ # inside contract
      if $F_{C'} < F_{p+1}$ **then** $x^* = x_{C'}$ **else** *shrink* = true
    end if
  end if
  if *shrink* == true **then**
    $x_i = x_1 + \varsigma(x_i - x_1)$, $i = 2,\ldots,p+1$
  else
    $x_{p+1} = x^*$
  end if
end while

The simplex adjusts to the contours of the objective function (i.e., it can 'stretch' itself) and so can make larger steps into favourable directions. But this flexibility can also be a disadvantage: try to visualise a narrow valley along which a long-stretched simplex  advances. If this valley were to take a turn, the simplex could not easily adapt (see [41] for a discussion). This phenomenon seems to occur in our problem. When we initialise the simplex, the maximum of a parameter value is 5% greater than its minimum, and this is true for all parameters by construction, see Equation (2.14). Thus the stretch in relative terms along any dimension is the same. When we run a search and compare this initial with the final simplex, we often find that the stretch along some dimensions is 200 times greater than along other dimensions; the condition number of a simplex often increases from 10^3 or so to 10^{12} and well beyond. This can be a warning sign, and here it is: it turns out that restarting the algorithm, i.e., re-initialising the simplex several times, leads to much better solutions.

2.3.4 Constraints

We constrain all heuristics to favour interpretable values: thus we want non-negative variances, correlation between -1 and 1, and parameters like κ, σ and λ also non-negative. These

constraints are implemented through penalty terms. For any violation a positive number proportional to the violation is added to the objective function. There are other approaches to incorporate constraints, for instance variable transformations (see [20] (Sect. 7.4) or [33]). Penalties have the advantage here that they are quick and easy to implement.

2.4 Experiments and Results

We create artificial data sets to test our techniques. The spot price S_0 is 100, the riskfree rate r is 2%, there are no dividends. We compute prices for strikes X from 80 to 120 in steps of size 2, and maturities τ of 1/12, 3/12, 6/12, 9/12, 1, 2 and 3 years. Hence our surface comprises $21 \times 7 = 147$ prices. Given a set of parameters, we compute option prices and store them as the true prices. Then we run 10 times each of our methods to solve problem (2.13) and see if we can recover the parameters; the setup implies that a perfect fit is possible. The parameters for the Heston model come from the following table (each column is one parameter set):

$\sqrt{v_0}$	0.3	0.3	0.3	0.3	0.4	0.2	0.5	0.6	0.7	0.8
$\sqrt{\theta}$	0.3	0.3	0.2	0.2	0.2	0.4	0.5	0.3	0.3	0.3
ρ	-0.3	-0.7	-0.9	0.0	-0.5	-0.5	0.0	-0.5	-0.5	-0.5
κ	2.0	0.2	3.0	3.0	0.2	0.2	0.5	3.0	2.0	1.0
σ	1.5	1.0	0.5	0.5	0.8	0.8	3.0	1.0	1.0	1.0

For the Bates model, we use the following parameter sets:

$\sqrt{v_0}$	0.3	0.3	0.3	0.3	0.4	0.2	0.5	0.6	0.7	0.8
$\sqrt{\theta}$	0.3	0.3	0.2	0.2	0.2	0.4	0.5	0.3	0.3	0.3
ρ	-0.3	-0.7	-0.9	0.0	-0.5	-0.5	0.0	-0.5	-0.5	-0.5
κ	2.0	0.2	3.0	3.0	0.2	0.2	0.5	3.0	2.0	1.0
σ	0.3	0.5	0.5	0.5	0.8	0.8	1.0	1.0	1.0	1.0
λ	0.1	0.1	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
μ_J	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1
σ_J	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

With ten different parameter sets for every model and with ten optimisation runs (restarts) for each parameter set, we have 100 results for each optimisation method. For each restart we store the value for the objective function (the mean percentage error; Equation (2.13)), and the corresponding parameter estimates. For the latter we compute absolute errors, i.e.,

$$\text{error} = |\,\text{estimated parameter} - \text{true parameter}\,|\,.$$

Below we look at the distributions of these errors.

All algorithms are coded in `Matlab`, for the direct search we use `Matlab's fminsearch`. We ran a number of preliminary experiments to find effective parameter values for the algorithms. For DE, the F-parameter should be set to around 0.3–0.5 (we use 0.5); very low or high values typically impaired performance. The CR-parameter had less influence, but levels close to unity worked best; each new candidate solution is then likely changed in many dimensions. For PS, the main task is to accelerate convergence. Velocity should not be

allowed to become too high, hence inertia should be below unity (we set it to 0.7); we also restricted maximum absolute velocity to 0.2. The stopping criterion for DE and PS is a fixed number of function evaluations (population size × generations); we run three settings,

$$\begin{aligned} 1\,250 \quad & (25\times 50)\,, \\ 5\,000 \quad & (50\times 100)\,, \\ 20\,000 \quad & (100\times 200)\,. \end{aligned}$$

On an Intel P8700 single core at 2.53 GHz with 2 GB of RAM one run takes about 10, 40 or 160 seconds, respectively. An alternative stopping criterion is to halt the algorithm once the diversity within the population – as measured for instance by the range of objective function or parameter values – falls below a tolerance level. This strategy works fine for DE where the solutions generally converge rapidly, but leads to longer run times for PS.

For the hybrid methods, we use a population of 25 solutions, and run 50 generations. Every 10 generations, one or three solutions are selected, either the best solutions ('elitists') or random solutions. These solutions are then used as the starting values of a local search. This search comprises repeated applications of Nelder–Mead, restricted to 200 iterations each, until no further improvement can be achieved; 'further improvement' is a decrease in the objective function greater than 0.1% . One run of the hybrid takes between 10 and 30 seconds.

2.4.1 Goodness-of-Fit

We first discuss the achieved objective function values, i.e., the average percentage pricing error. Figures 2.9 to 2.16 show the empirical distributions of the pricing errors. For the pure population-based approaches, the grey scales indicate the increasing number of function evaluations (1 250, 5 000, and 20 000).

For the Heston model, all methods converge quickly to very good solutions with pricing errors less than one percent; we can also achieve a perfect fit. (Not reported: we ran further tests for DE and PS with more function evaluations which increased running time to 3–5 minutes; then both algorithms converged on the true solutions without exception. However, for practical purposes, the precision achieved here is sufficient.) DE converges faster than PS. This becomes apparent for the hybrid algorithms. Here, for DE it makes little difference how we select solutions for local search (random or elitists) because all members of the population are similar. For PS, selecting solutions by quality works better, see Figures 2.11 and 2.12.

It is more difficult to calibrate the Bates model. Figures 2.13 to 2.16 show that convergence is slower here. The hybrid methods perform very well; in particular so when based on DE, or on PS with solutions chosen by quality. For both the Heston and the Bates model, the hybrid performed best, with a small advantage for the DE-based algorithm.

2.4.2 Parameters

It is also of interest to know how fast the parameter estimates converge to their true values – or if they do. To save space here, we only present results for DE; convergence for other methods looked similar. Figures 2.17 and 2.18 show that for the Heston model, the parameters converge roughly in-line with the objective function; see for instance Figure 2.9. Still, good fits (say, less than one percent pricing error) can be achieved with quite different parameter values.

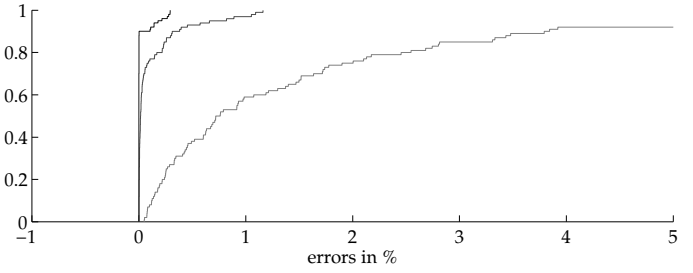


Fig. 2.9 Heston model: Distributions of errors for Differential Evolution. Darker grey indicates more iterations.

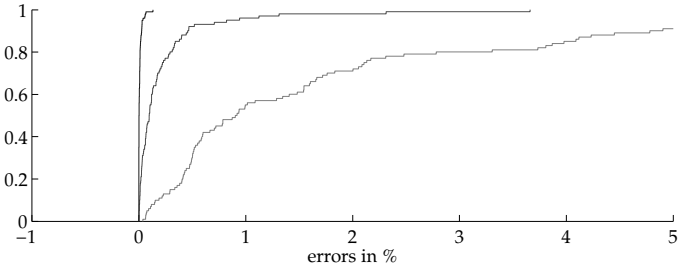


Fig. 2.10 Heston model: Distributions of errors for Particle Swarm Optimisation. Darker grey indicates more iterations.

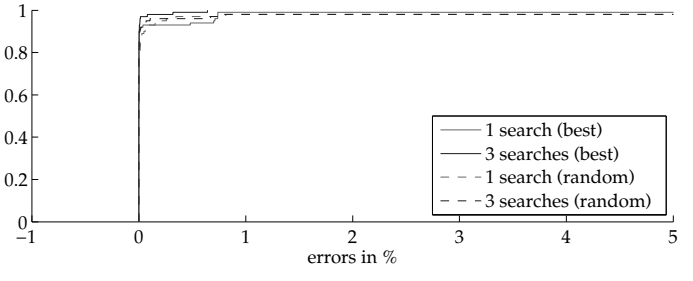


Fig. 2.11 Heston model: Distributions of errors for hybrid based on Differential Evolution

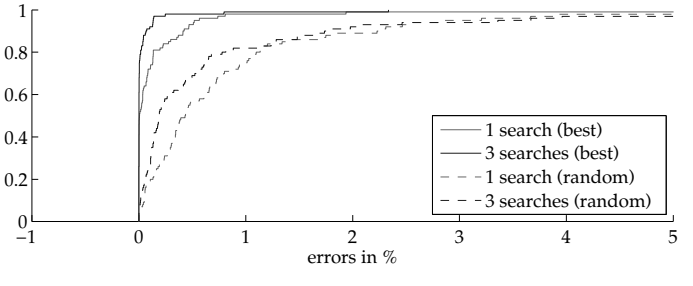


Fig. 2.12 Heston model: Distributions of errors for hybrid based on Particle Swarm Optimisation

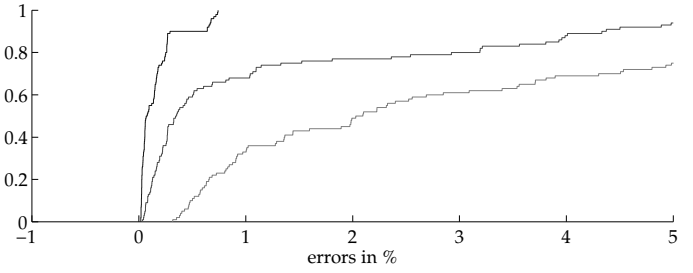


Fig. 2.13 Bates model: Distributions of errors for Differential Evolution. Darker grey indicates more iterations.

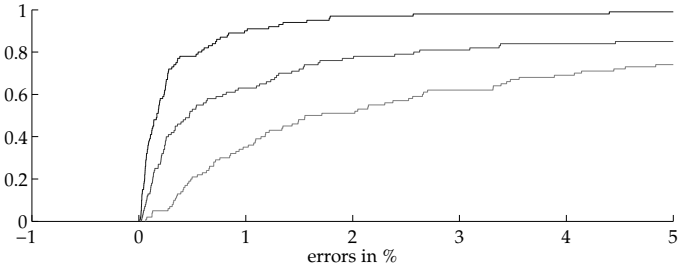


Fig. 2.14 Bates model: Distributions of errors for Particle Swarm Optimisation. Darker grey indicates more iterations.

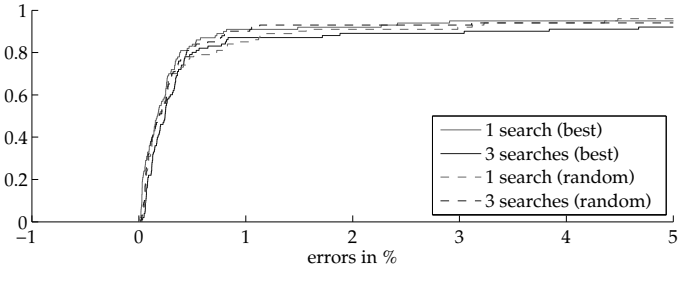


Fig. 2.15 Bates model: Distributions of errors for hybrid based on Differential Evolution

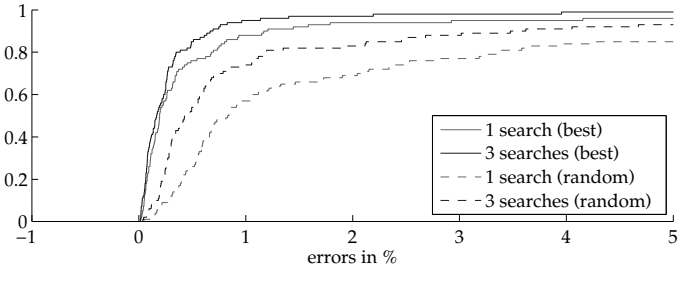


Fig. 2.16 Bates model: Distributions of errors for hybrid based on Particle Swarm Optimisation

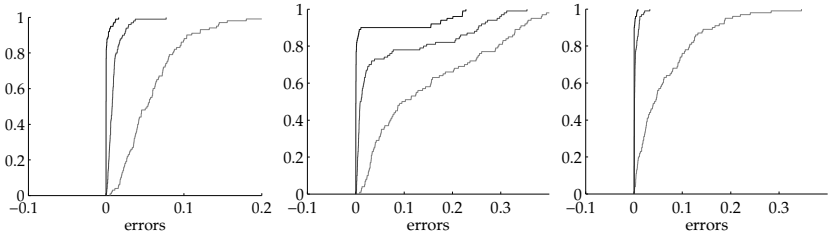


Fig. 2.17 Convergence of parameter estimates for the Heston model with DE. The figure shows the distributions of absolute errors in the parameters for $\sqrt{v_0}$ (left), $\sqrt{\theta}$ (middle), and ρ (right).

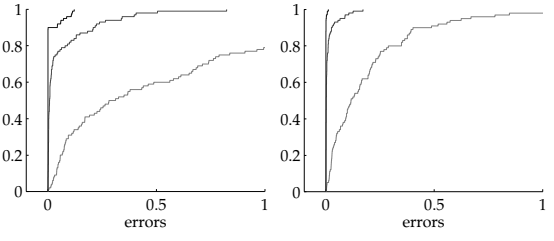


Fig. 2.18 Convergence of parameter estimates for the Heston model with DE. The figure shows the distributions of absolute errors in the parameters for κ (left), and σ (right).

For the Bates model, the results are worse: Figures 2.19, 2.20, and 2.21 give results. Those parameters that are also in the Heston model are estimated less precisely; but for the jump parameters (λ, μ_J and σ_J) there is essentially no convergence. No convergence in the parameters does not mean we cannot get a good fit; compare Figures 2.13 to 2.16 and the discussion below. This non-convergence is to some extent owed to our choice of parameters. Experiments with Merton's model (not reported) showed that for 'small' mean jumps μ_J of magnitude -10% or -20%, it is difficult to recover parameters precisely because many parameter values give price errors of practically zero. In other words, the optimisation is fine, we can well fit the prices, but we cannot accurately identify the different parameters. In any case, parameter values of the magnitude used here have been reported in the literature (e.g., [14] or [35]). The precision improves for large jumps. This is consistent with other studies: [25] for instance report relatively-precise estimates for a mean jump size -90%. At some point, this begs the question how much reason we should impose on the parameters. An advantage of theoretical models over simple interpolatory schemes is the interpretability of parameters. If we can only fit option prices by unrealistic parameters, there is little advantage in using such models.

The convergence of parameters is also pictured in Figures 2.22 to 2.26. The graphics show the parameter errors of a solution compared with the overall price error of the respective solutions. In the middle panel of Figure 2.22, for example, we see that in the Heston model we can easily have a price error of less than one percent, but a corresponding error in long-run volatility of 0.2 (i.e., 20 percentage points!). Most remarkable is Figure 2.26: we see that for μ_J and σ_J, practically any value can be compatible with a low price error.

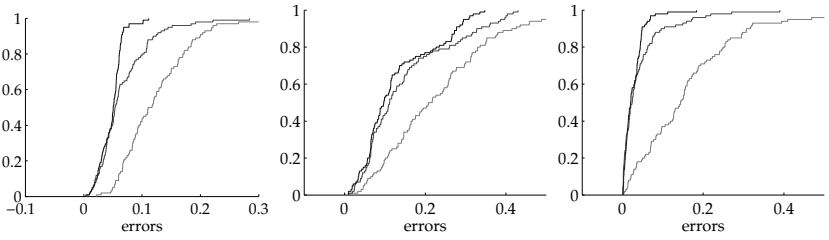


Fig. 2.19 Convergence of parameter estimates for the Bates model with DE. The figure shows the distributions of absolute errors in the parameters for $\sqrt{v_0}$ (left), $\sqrt{\theta}$ (middle), and ρ (right).

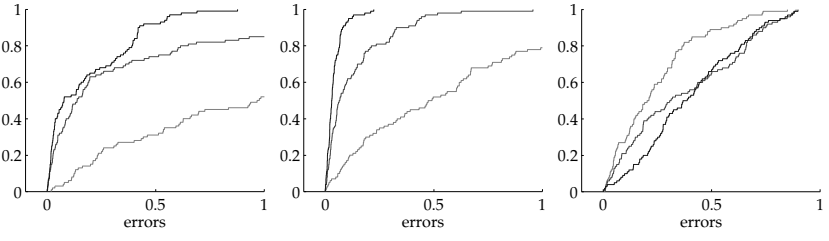


Fig. 2.20 Convergence of parameter estimates for the Bates model with DE. The figure shows the distributions of absolute errors in the parameters for κ (left), σ (middle), and λ (right).

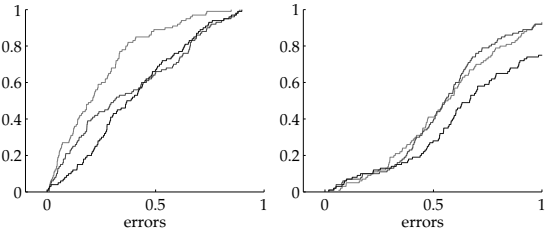


Fig. 2.21 Convergence of parameter estimates for the Bates model with DE. The figure shows the distributions of absolute errors in the parameters for μ_J (left), and σ_J (right).

2.4.3 Conditioning of the Problem

We have seen that good results in terms of the objective function – i.e., low price errors – can go along with sometimes large errors in the estimated parameters. This can have to do with specific parameter settings as discussed above for Merton's jump diffusion model; it also reflects the fact that both stochastic volatility and jumps can generate the volatility smile (though stochastic volatility models are better at this for long maturities, and jump models are better for short expirations, see [11]). The optimisation procedure hence cannot clearly attribute the smile to either cause. This is an identification problem, a problem of the model, not of the numerical technique.

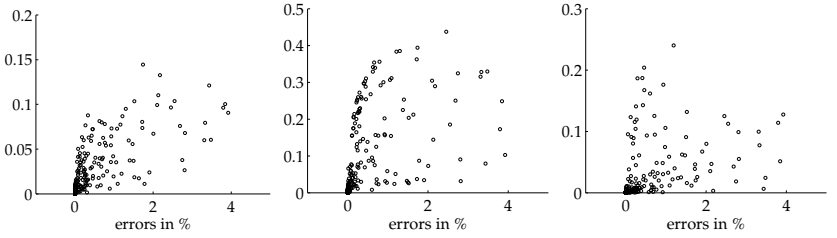


Fig. 2.22 Goodness of fit vs parameter errors for the Heston model with DE. The x-scale gives the objective function value (average error in percentage points) for solutions. The y-scale shows the associated absolute errors in the parameters for $\sqrt{v_0}$ (left), $\sqrt{\theta}$ (middle), and ρ (right).

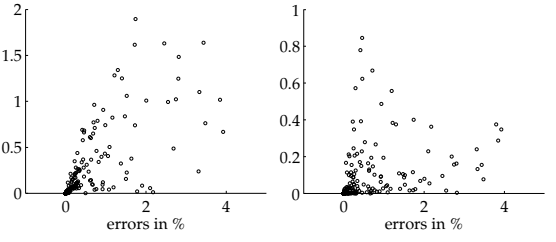


Fig. 2.23 Goodness of fit vs parameter errors for the Heston model with DE. The x-scale gives the objective function value (average error in percentage points) for solutions. The y-scale shows the associated absolute errors in the parameters for κ (left), and σ (right).

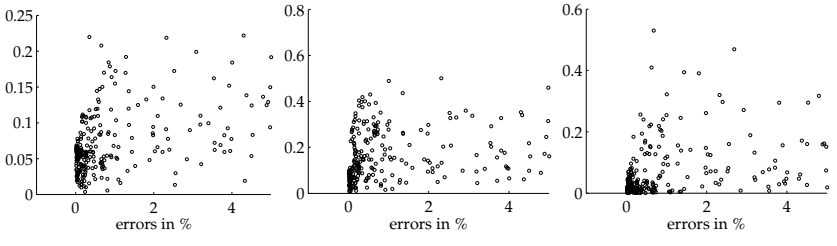


Fig. 2.24 Goodness of fit vs parameter errors for the Bates model with DE. The x-scale gives the objective function value (average error in percentage points) for solutions. The y-scale shows the associated absolute errors in the parameters for $\sqrt{v_0}$ (left), $\sqrt{\theta}$ (middle), and ρ (right).

Our objective function (2.13) can be rewritten as a system of nonlinear equations

$$\frac{\left|C_i^{\text{model}} - C_i^{\text{market}}\right|}{C_i^{\text{market}}} = 0 \tag{2.15}$$

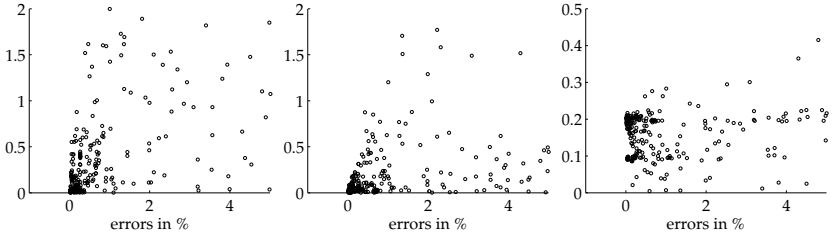


Fig. 2.25 Goodness of fit vs parameter errors for the Bates model with DE. The *x*-scale gives the objective function value (average error in percentage points) for solutions. The *y*-scale shows the associated absolute errors in the parameters for κ (left), σ (middle), and λ (right).

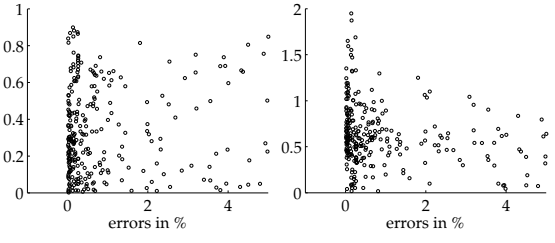


Fig. 2.26 Goodness of fit vs parameter errors for the Bates model with DE. The *x*-scale gives the objective function value (average error in percentage points) for solutions. The *y*-scale shows the associated absolute errors in the parameters for μ_J (left), and σ_J (right).

where $i \in 1,\ldots,M$. The number of market prices M is greater than the number of parameters, so the system is overidentified; it can only be solved by minimising a norm of the residual. The conditioning of a system of equations does not necessarily affect the size of the residual: even badly-conditioned equations may result in small residuals; but the parameters can then not be accurately determined. This seems to be the case here.

To test the conditioning, we explicitly evaluate the Jacobian matrix $\mathscr{J}$ at every step of the optimisation. $\mathscr{J}$ is a matrix of size $M \times \#\{x_0\}$ where $\#\{x_0\}$ is number of parameters of a solution x_0, h is a small number, and e_j is the jth unit vector. We denote c_0 a vector of length M that stores the relative price errors. Algorithm 2.6 describes how to approximate the Jacobian of (2.15) by a forward difference.

Algorithm 2.6 Computing the Jacobian matrix

1: set h
2: compute c_0: the left-hand side of Equation (2.15) for parameters x_0
3: **for** $j = 1$ **to** $\#\{x_0\}$ **do**
4: compute $x_j = x_0 + he_j$
5: compute c_j: the left-hand side of Equation (2.15) for parameters x_j
6: $\mathscr{J}_{\cdot,j} = (c_j - c_0)/h$
7: **end for**
8: compute condition number of $\mathscr{J}$

We find that mostly the condition number of the models is numerically acceptable for double precision (say, of order 10^5 or 10^6), even though there are steps where the conditioning deteriorates dramatically. An example for the Bates model ($\sqrt{v_0} = 0.3$, $\sqrt{\theta} = 0.3$, $\rho = -0.3$, $\kappa = 2$, $\sigma = 0.3$, $\lambda = 0.10$, $\mu_J = -0.10$, $\sigma_J = 0.10$) is given in Figure 2.27. For the Heston model, the conditioning is better.

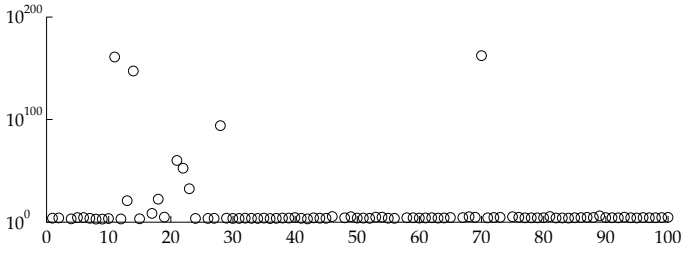


Fig. 2.27 Condition number of Jacobian for one solution over 100 generations

Just because the condition number is numerically acceptable does not mean the model is fine. For intuition: in a linear regression model, the Jacobian is the data matrix. The condition number of this matrix can be acceptable even though high correlation between the columns may prohibit any sensible inference. The following `Matlab` script sets up a linear regression model with extremely correlated regressors.

```
% set number of observations, number of regressors
nObs = 150; nR = 8;

% set up correlation matrix
C = ones(nR,nR) * 0.9999; C( 1:(nR+1):(nR*nR) ) = 1;

% create data
X = randn(nObs,nR); C = chol(C); X = X*C; bTrue = randn(nR,1);
y = X*bTrue + randn(nObs,1)*0.2;
```

The regression `X\y` can be computed without numerical problems, though we had better not interpret the coefficients.

The conditioning of the Bates model does not matter much if we just aim to interpolate current option prices; it is a problem if we want to compute parameters accurately from option prices – which we cannot do for this model.

2.5 Conclusion

In this chapter we have investigated the calibration of option pricing models. We have shown how to calibrate the parameters of a model with heuristic techniques, Differential Evolution and Particle Swarm Optimisation, and that we can improve the performance of these methods by adding a Nelder–Mead direct search. While good price fits could be achieved with all

methods, the convergence of parameter estimates was much slower; for the jump parameters of the Bates model, there was no convergence. This, it must be stressed, is not a problem of the optimisation technique, but it stems from the model. In comparison, parameters of the Heston model could be estimated more easily.

In empirical studies on option pricing models (e.g., [3], or [35]), the calibration is often taken 'for granted'; it is rarely discussed whether, for instance, restarts of an optimisation routine with different starting values would have resulted in different parameter estimates, and how such different estimates would have had influenced the studies' results. In [21] we showed that standard gradient-based methods often fail for the kinds of calibration problems discussed in this chapter, and that restarts with different starting values can lead to very different solutions. Different parameter values may lead to good overall fits in terms of prices, but these different parameters may well imply very different Greeks, or have a more marked influence on prices of exotic options. Hence empirical studies that look for example into hedging performance should take into account the sensitivity of their results with respect to calibration. With luck, all that is added is another layer of noise; but the relevance of optimisation is to be investigated by empirical testing, not by conjecturing. Such testing is straightforward: we just need to rerun our empirical tests many times, each time also rerunning our calibration with alternative starting values, and hence can get an idea of the sensitivity of outcomes with respect to optimisation quality. Ideally, optimisation quality in-sample should be evaluated jointly with empirical, out-of-sample performance of the model, see [22].

Our findings underline the point raised in [23] that modellers in quantitative finance should be sceptical of purely-numerical precision. Model risk is a still under-appreciated aspect in quantitative finance (and one that had better not be handled by rigorous mathematical modelling). For instance, [35] showed that the choice of an option pricing model can have a large impact on the prices of exotic options, even though all models were calibrated to the same market data. Unfortunately, different calibration criteria lead to different results, see [14]. In the same vein, [27] find that different models, when calibrated to plain vanilla options, exhibit widely-differing pricing performance when used to explain actual prices of barrier options. Our results suggest that the lowly numerical optimisation itself can make a difference. How important this difference is needs to be assessed empirically.

Acknowledgements. Both authors gratefully acknowledge financial support from the EU Commission through MRTN-CT-2006-034270 COMISEF; they would like to thank Benoît Guilleminot for many discussions on the subject.

References

1. Albrecher, H., Mayer, P., Schoutens, W., Tistaert, J.: The Little Heston Trap, pp. 83–92. Wilmott (2007)
2. Bakshi, G., Madan, D.B.: Spanning and Derivative-Security Valuation. Journal of Financial Economics 55(2), 205–238 (2000)
3. Bakshi, G., Cao, C., Chen, Z.: Empirical Performance of Altemative Option Pricing Models. Journal of Finance 52(5), 2003–2049 (1997)
4. Bates, D.S.: Jumps and Stochastic Volatility: Exchange Rate Processes Implicit in Deutsche Mark Options. Review of Financial Studies 9(1), 69–107 (1996)
5. Black, F., Scholes, M.: The Pricing of Options and Corporate Liabilities. Journal of Political Economy 81(3), 637–654 (1973)

6. Carr, P., Madan, D.B.: Option Valuation Using the Fast Fourier Transform. Journal of Computational Finance 2(4), 61–73 (1999)
7. Chourdakis, K.: Option Pricing Using The Fractional FFT. Journal of Computational Finance 8(2), 1–18 (2005)
8. Coleman, T.F., Li, Y., Verma, A.: Reconstructing the Unknown Local Volatility Function. Journal of Computational Finance 2(3), 77–102 (1999)
9. Cont, R., Tankov, P.: Non-parametric Calibration of Jump-diffusion Option Pricing Models. Journal of Computational Finance 7(3), 1–49 (2004)
10. Crépey, S.: Calibration of the Local Volatility in a Trinomial Tree Using Tikhonov Regularization. Inverse Problems 19(1), 91–127 (2003)
11. Das, S.R., Sundaram, R.K.: Of Smiles and Smirks: A Term Structure Perspective. The Journal of Financial and Quantitative Analysis 34(2), 211–239 (1999)
12. Davis, P.J., Rabinowitz, P.: Methods of Numerical Integration, 2nd edn. Dover, New York (2007)
13. Derman, E., Kani, I.: The Volatility Smile and Its Implied Tree. Goldman Sachs Quantitative Strategies Research Notes (1994)
14. Detlefsen, K., Härdle, W.K.: Calibration Risk for Exotic Options. Journal of Derivatives 14(4), 47–63 (2007)
15. Duffie, D., Pan, J., Singleton, K.: Transform Analysis and Asset Pricing for Affine Jump-Diffusions. Econometrica 68(6), 1343–1376 (2000)
16. Dupire, B.: Pricing with a Smile. Risk 7(1), 18–20 (1994)
17. Eberhart, R.C., Kennedy, J.: A new optimizer using particle swarm theory. In: Proceedings of the Sixth International Symposium on Micromachine and Human Science, Nagoya, Japan, pp. 39–43 (1995)
18. Fang, F., Oosterlee, C.: A Novel Pricing Method for European Options Based on Fourier-Cosine Series Expansions. SIAM Journal on Scientific Computing 31(2), 826–848 (2008)
19. Gatheral, J.: The Volatility Surface. Wiley, Chichester (2006)
20. Gill, P.E., Murray, W., Wright, M.H.: Practical Optimization. Elsevier, Amsterdam (1986)
21. Gilli, M., Schumann, E.: Calibrating the Heston Model with Differential Evolution. In: Di Chio, C., Brabazon, A., Di Caro, G.A., Ebner, M., Farooq, M., Fink, A., Grahl, J., Greenfield, G., Machado, P., O'Neill, M., Tarantino, E., Urquhart, N. (eds.) EvoApplications 2010, Part II. LNCS, vol. 6025, pp. 242–250. Springer, Heidelberg (2010a)
22. Gilli, M., Schumann, E.: Optimal Enough? Journal of Heuristics (2010b) (forthcoming)
23. Gilli, M., Schumann, E.: Optimization in Financial Engineering – An essay on 'good' solutions and misplaced exactitude. Journal of Financial Transformation 28, 117–122 (2010c)
24. Hale, N., Trefethen, L.N.: New quadrature formulas from conformal maps. SIAM Journal of Numerical Analysis 46(2), 930–948 (2008)
25. He, C., Kennedy, J., Coleman, T.F., Forsyth, P.A., Li, Y., Vetzal, K.: Calibration and Hedging under Jump Diffusion. Review of Derivatives Research 9(1), 1–35 (2006)
26. Heston, S.L.: A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bonds and Currency options. Review of Financial Studies 6(2), 327–343 (1993)
27. Jessen, C., Poulsen, R.: Empirical Performance of Models for Barrier Option Valuation. Tech. rep. University of Copenhagen (2009), `http://www.math.ku.dk/~rolf/papers.html`
28. Kilin, F.: Accelerating the Calibration of Stochastic Volatility Models. In: Centre for Practical Quantitative Finance Working Paper Series, vol. 6 (2007)

29. Lewis, A.L.: Option Valuation under Stochastic Volatility. Finance Press (2000)
30. Merton, R.C.: Option Pricing when Underlying Stock Returns are Discontinuous. Journal of Financial Economics 3(1-2), 125–144 (1976)
31. Moscato, P.: On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts – Towards Memetic Algorithms. Tech. Rep. 790, CalTech. California Institute of Technology (1989)
32. Nelder, J.A., Mead, R.: A Simplex Method for Function Minimization. Computer Journal 7(4), 308–313 (1965)
33. Powell, M.: Problems Related to Unconstrained Optimization. In: Murray, W. (ed.) Numerical Methods for Unconstrained Optimization. Academic Press, London (1972)
34. Schoutens, W.: Lévy Processes in Finance: Pricing Financial Derivatives. Wiley, Chichester (2003)
35. Schoutens, W., Simons, E., Tistaert, J.: A Perfect Calibration! Now What? Wilmott (2004)
36. Spendley, W., Hext, G., Himsworth, F.: Sequential Application of Simplex Designs in Optimisation and Evolutionary Operation. Technometrics 4(4), 441–461 (1962)
37. Storn, R.M., Price, K.V.: Differential Evolution – a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. Journal of Global Optimization 11(4), 341–359 (1997)
38. Talbi, E.G.: A Taxonomy of Hybrid Metaheuristics. Journal of Heuristics 8(5), 541–564 (2002)
39. Trefethen, L.N.: Is Gauss Quadrature Better than Clenshaw–Curtis? SIAM Review 50(1), 67–87 (2008)
40. Winker, P., Gilli, M.: Applications of Optimization Heuristics to Estimation and Modelling Problems. Computational Statistics and Data Analysis 47(2), 211–223 (2004)
41. Wright, M.H.: Direct search methods: Once scorned, now respectable. In: Griffiths, D., Watson, G. (eds.) Numerical Analysis 1995 (Proceedings of the 1995 Dundee Biennial Conference in Numerical Analysis), pp. 191–208. Addison Wesley Longman, Amsterdam (1996)

3

A Comparison between Nature-Inspired and Machine Learning Approaches to Detecting Trend Reversals in Financial Time Series

Antonia Azzini[1], Matteo De Felice[2,3], and Andrea G.B. Tettamanzi[1]

[1] Università degli Studi di Milano, Dipartimento di Tecnologie dell'Informazione, Italy
{antonia.azzini,andrea.tettamanzi}@unimi.it
[2] ENEA (Italian Energy New Technology and Environment Agency)
[3] Università degli Studi "Roma Tre", Dipartimento di Informatica e Automazione, Italy
defelice@dia.uniroma3.it

Summary. Detection of turning points is a critical task for financial forecasting applications. This chapter proposes a comparison between two different classification approaches on such a problem. Nature-Inspired methodologies are attracting growing interest due to their ability to cope with complex tasks like classification, forecasting, and anomaly detection problems. A swarm intelligence algorithm, namely Particle Swarm Optimization (PSO), and an artificial immune system algorithm, namely Negative Selection (NS), have been applied to the task of detecting turning points, modeled as an Anomaly Detection (AD) problem. Particular attention has also been given to the choice of the features considered as inputs to the classifiers, due to the significant impact they may have on the overall accuracy of the approach. In this work, starting from a set of eight input features, feature selection has been carried out by means of a greedy hill climbing algorithm, in order to analyze the incidence of feature reduction on the global accuracy of the approach. The performances obtained from the two approaches have also been compared to other traditional machine learning techniques implemented by WEKA and both methods have been found to give interesting results with respect to traditional techniques.

3.1 Introduction

Every financial time series follows a zig-zag path, consisting of alternating upward and downward price swings [18]. Such a behaviour happens and can be observed at all scales [26], even if price movements are not regular and look unpredictable. A typical financial market action consists of alternating up- and down-trends, separated by turning points, which correspond to local maxima or minima of the prices of a financial instrument. One of the most important objectives of financial market forecasting techniques is to detect turning points consistently and correctly. Such price movements are not regular and the application of common forecasting tools does not provide satisfactory results.

Several studies in the literature recognize Technical Analysis as one of the most popular approaches to analyze price movements. Indeed, based on the Dow Theory principles [25], it

A. Brabazon et al. (Eds.): Natural Computing in Comput. Finance, SCI 380, pp. 39–59.
springerlink.com

uses historical prices to predict future movements, reformulating the approach into a classification and pattern recognition problem, that attempts to classify observations into categories, generally by using pattern learning [23, 36]. Researches in this area involved the detection of patterns and the definition of 'positive' or 'negative' classes of data.

In this sense, the task of predicting the overall price movements of a security in the next period can be modeled as an anomaly detection problem. In behaviour-based approaches to anomaly detection, the model of normal behaviour is constructed from an observed sample of normally occurring patterns.

In financial problems, a normal pattern of behaviour for the price of a security may be defined as a continuing trend, whereas an anomaly in this setting would consist of a trend reversal. Then, the problem of anomaly detection can be naturally recast into a 'one-class' classification problem, where only examples of a single class are used and all the others are considered to be outliers. In this research area, machine learning and nature-inspired algorithms appear to be promising, being able to discover satisfactory solutions with unbalanced datasets (the number of instances of normal behaviour is, by definition, greater than the number of anomalies). In this sense particular attention is given to the ideas of the Artificial Immune Systems (AIS), computational systems inspired by theoretical immunology.

The biological immune system is a robust, complex, adaptive system that defends the body from foreign pathogens. Indeed, a fundamental problem solved by most AIS can be thought of as learning to discriminate between 'self' (the normally occurring patterns in the system being protected, e.g., the body) and 'non-self' (foreign pathogens, such as bacteria or viruses, or components of self that are no longer functioning normally). Their development and application domain are related to those of other soft computing paradigms such as Artificial Neural Networks (ANNs), Evolutionary Algorithms (EAs) and Fuzzy Systems (FS). Actually, the AIS literature boasts successful applications in a wide variety of areas, such as optimization, data analysis, computer security, pattern recognition and classification.

Nature-inspired techniques have been demonstrated to be effective for financial applications in some fields [30] and hybridized approaches with classical machine learning methods were proposed, especially in particularly complex applications [22]. The problem of predicting the trend of a financial instrument is approached using several methods, including, Particle Swarm Optimization [7], which we adopted as well in our approach, and, in a large number of cases, Artificial Neural Networks, often focusing on the selection of the technical indices to be used for the statistical analysis [19, 40].

In this chapter, we describe a more complete and in-depth comparative analysis of two approaches presented for the first time in [3], where we employed two types of nature-inspired algorithms, respectively Particle Swarm Optimization and Negative Selection (taken from AIS), applied to the creation of an optimal hyperspherical detector set. The former algorithm leads to a positive selection mechanism whereas the latter uses a negative selection process, defining the anomalous space as the complement of the normal one. The performances of the two approaches are compared with the aim of assessing which one is more suited for trend reversal detection.

Particular attention should also be given to the choice of the features considered as inputs to the classifiers, due to the significant impact that they may have over the global accuracy of the approach. In this work, starting from a set of eight input features, a feature selection procedure (a greedy algorithm) has been carried out manually, in order to analyze in detail the incidence of feature reduction on the global accuracy of the approach.

This paper is organized as follows. Section 3.2 presents the financial problem. The detector models and the other machine learning approaches considered for comparison are stated in Section 3.3, while a discussion about the incidence of the feature reduction is presented in

Section 3.4, together with the two feature selection algorithms considered for this scope. The results obtained from the experiments carried out by applying PSO and NS algorithms are presented in Section 3.5. Moreover, to make the case for the attractiveness of the approach, a detailed comparison of the classification performances obtained by using some of machine learning algorithms implemented in WEKA [38] has been provided, while Section 3.6 discusses the results obtained. Finally, Section 3.7 provides some concluding remarks.

3.2 Problem Description

As a general principle, wherever there is an oscillation or a vibration, whether in the tides of an ocean or in the heat of a system or in a pendulum, it should be possible to extract some energy by means of an implement that acts so as to damp the oscillation. If the mass or energy of the oscillating system were very large with respect to the energy-extracting implement, the damping caused by energy extraction would be negligible and the process could continue indefinitely.

As previously indicated, even the most casual observer of a financial time series will notice that prices of financial instruments move up and down [18]; furthermore, this behaviour happens and can be observed at all scales [26]. Therefore, a trader on the financial markets should be able to extract a profit from the oscillations in the price of a security, much like one should be able to extract energy from an oscillating physical system, the liquidity of a security being the financial notion corresponding to the mass of the system under exploitation.

An obstacle to putting this idea into practice is that the price movements of real securities traded on financial markets are not regular and look, at least to some extent, unpredictable. However, if an algorithm is able to predict with a less than 50% error the overall direction of price movement in the next period, an arbitrary profit could be extracted from trading a very 'liquid' security, provided that a sound money management strategy is enforced [37].

We equate the task of predicting the overall price movement of a security in the next period to the task of detecting an anomaly in a system. More specifically, a normal pattern of behaviour for the price of a security may be defined as continuing a trend, whereas an anomaly consists, in this setting, of a trend reversal.

To demonstrate the viability of such an idea, we approach here a very simple instance of the above general problem, where daily series of an index are considered and a reversal is defined as a change in the sign of the log return of the next period with respect to the previous period. In this sense we are able to define, for each day i of the time series, the target of our detection problem with a value equal to 0 or 1, corresponding, respectively, to a normal behaviour defined as continuing a trend, or an anomaly, when a trend reversal occurs. The target setting is defined by Equation 3.1:

$$\text{target}(t) = \begin{cases} 0 & \text{if} \operatorname{sgn}\left(\log\left(\frac{C(t)}{C(t-1)}\right)\right) = \operatorname{sgn}\left(\log\left(\frac{C(t+1)}{C(t)}\right)\right), \\ 1 & \text{otherwise}, \end{cases} \tag{3.1}$$

where $C(t)$ represents the closing price of the financial instrument at time t.

The problem inputs are then defined, to begin with, by considering the log return of the daily historical prices and 7 different technical indicators for the same financial instrument. Such technical indicators correspond to some of the most popular indicators used in technical analysis and summarize important features of the time series of the considered financial instrument [10]. They also represent useful statistics and technical information, and typically they

are used for modeling some aspects of price movements. Some indicators, for example, use only closing prices, while others incorporate the volume of trades into their formulae [23].

A brief description of the indicators considered in this work is reported in Table 3.1. Our work discusses the S&P 500 index and Figure 3.1 shows the distribution of the eight indicators for the period we have taken into account (from April 1992 to September 2009).

Table 3.1 Input Technical Indicators

Index	Input Technical Indicator	Description
X1	r_i	Log Return
X2	MA50(i)	50-day Moving Average
X3	EMA50(i)	50-day Exponential Moving Average
X4	RSI(i)	Relative Strength Index
X5	Momentum(i)	Rate of Price Change
X6	ROC(i)	Rate Of Change
X7	MACD(i)	Moving Average Convergence/Divergence
X8	SIGNAL(i)	Exponential Moving Average on MACD

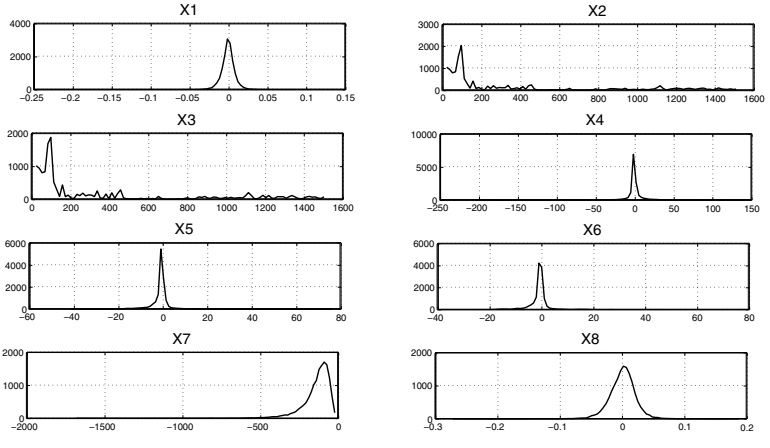


Fig. 3.1 Distribution of the input features

There are several different technical indicators used by practitioners, and the choice of those selected for this approach was made, at the beginning, also by selecting, together with the most widely used, the most representative of different categories (e.g. moving averages, oscillators). Nevertheless, due to the high computational cost required to evaluate the performance of the approach in an eight-dimensional space, it was interesting to analyze the incidence of the input reduction in practice. For this reason we 'manually' implemented a kind of feature selection through a 'greedy' algorithm, in order to find an optimal subset of inputs, capable of giving results of comparable quality as when using all eight features, thus reducing the overall computational efforts. This algorithm has been compared with a solution

based on an exhaustive search has also been implemented, but it has been restricted to the reduction of just two features, for considering the elimination of more features would have required too much computational effort. This topic of input reduction and the outcome of its application to the problem at hand are discussed in detail below, in Section 3.4.

3.3 Anomaly Detection Methods

When we are coping with a classification problem of 'normal' vs. 'anomalous', having two characteristic and representative training sets that resemble the true data distribution might be hard or almost impossible and this problem becomes more pronounced when there is a small availability of training data. In this case we might consider only one class (data resembling normal behaviour) and define the outlier points as 'anomalous' behaviour. Many models for one-class classification have been proposed, e.g., density methods, support vector machines (SVM), clustering algorithms [35].

Our main goal is to obtain a set of detectors covering the feature space in the best way for this particular anomaly detection problem. In this work, we considered different kinds of models to define detectors. Some of them are based on machine learning approaches, using for example different kinds of neural networks (MLPs, RBFNNs), nearest neighbour, decision trees, etc. Others are based on geometrical solutions like hyperspherical detectors, that consist of a set of coordinates, representing a point in the n-dimensional feature space, and a radius value (hyperradius).

In this chapter we consider sets of hyperspherical detectors as defined in [2], working in an n-dimension space. In particular, we use a vector-based representation where each detector d is described as follows:

$$d = (x_1, \ldots, x_n, \quad r), \tag{3.2}$$

with $x_1, \ldots, x_n$ the coordinates in the n-dimensional space and r the radius. Here, an observation (i.e., a point in the feature space) is considered non-anomalous if the distance (or similarity) between the point and the detector coordinates is less than or equal to the radius. As a measure of distance (or similarity), Euclidean distance will be used. The main advantage of this detector type is the ease of representation and implementation.

Models of normal behaviour can represent either the set of allowed patterns (positive detection) or the set of anomalous patterns (negative detection). This allows us to study the financial price movements either with a positive selection algorithm, as the Particle Swarm Optimization (PSO), or a negative one with the negative selection algorithm. Their basic steps are briefly reported in the following subsections.

All the considered approaches follow the commonly accepted practice of machine learning by defining a training and a test set, used, respectively, to create the detector sets and to test their generalization capabilities, even if according to two main differences. Indeed, by using PSO and NS, the training set contains only normal cases, while anomalies are added to the test set. On the other hand, in machine learning approaches, both training and test sets contain a balanced distribution of normal and anomalous cases.

The financial instrument considered in this work is the S&P 500 index, and its overall behaviour is represented by putting all the points of the time-series period into an n-dimensional space, where n varies according to the number of features considered. The hyperspherical detectors found by each method throughout the training set will then cover, respectively, the normal or the anomalous behaviours in that space during the test phase.

3.3.1 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a population-based optimization technique originally introduced by Eberhart and Kennedy [21]. In this algorithm, based on the simulation of the social behaviour of bird flocks and insects swarms, each individual (or particle), represented by a real-valued vector, moves in the solution space following two main attraction points: the best solution it has encountered so far and the best solution found by any other individual in the swarm.

Each particle represents a solution s_i and it has a velocity v_i which is updated at each iteration k of the algorithm as follows:

$$v_i^{k+1} = w v_i^k + c_1 R_1 \times (P_i^{\text{BEST}} - s_i^k) + c_2 R_2 \times (G^{\text{BEST}} - s_i^k) \tag{3.3}$$

with $w \in [0,1]$ an inertia factor, $c_1, c_2 \in \mathbb{R}$ two weight coefficients, R_1 and R_2 two uniformly distributed random numbers in $[0,1]$, P_i^{BEST} the best solution found so far by the particle i and G^{BEST} the best solution found by all the particles within the swarm. Then, the particle solutions are updated using $s_i^{k+1} = s_i^k + v_i^{k+1}$.

The algorithm implemented follows the common steps of a PSO algorithm, with the parameters defined in Table 3.6. The inertia weight varies and is reduced during the run, in order to have a good trade-off between a global (at the beginning) and a local exploration.

The PSO algorithm has been applied to several optimization problems [27], in particular there are various applications on classification and clustering where PSO turns out to be competitive compared to other classical machine learning algorithms [12, 33].

We used the algorithm to create a set of detectors that are able to maximize the coverage of the normal behaviours in the training phase and to minimize the radius of each detector. We implemented an algorithm that, for each iteration, removes from the dataset all the points covered by the best detector found in that iteration. This way, at the end of the algorithm, we obtain a set of detectors, each of them covering a subset of points in a satisfactory way. Maximum and minimum values are also defined for the radius of each particle, in order to avoid disruptive effects on the detection of unseen points in the space. The fitness function of particles in the PSO algorithm is given by Equation 3.4:

$$f(\mathbf{x}, r) = \frac{n^d}{r^d}, \tag{3.4}$$

where $\mathbf{x}$ is the vector with the coordinates in the solution space of the detector, r is the radius, n is the number of points covered by the detector, and d is the number of dimensions of the solution space.

The entire process of the PSO is reported in Algorithm 3.1. All the detectors in the `detectors_set` are used, and a point is defined normal if it is covered by, at least, one detector.

3.3.2 Negative Selection

The negative selection (NS) algorithm generates a set of detectors by first randomly creating candidates and then discarding those that correspond to the notion of self [13]. These detectors can later be used for detecting anomalies. In the original proposal, the NS has three basic steps: the *self definition*, defining a collection of binary strings S of limited length that needs protection or monitoring; the *detector generation*, a set of detectors that fail to match any strings in S; the *monitoring* phase, that monitors the set S for changes by continually matching the detectors; if any detector ever matches S, then an anomaly must have occurred, since

Algorithm 3.1 Particle Swarm Optimization Algorithm for Detectors Optimization

```
1: dataset ← load_dataset()
2: not_covered_points ← size(dataset)
3: detectors_set ← ∅
4: while not_covered_points > 0 do
5:     BEST_DETECTOR ← PSO_ALGORITHM(dataset, particles_number,
       max_iterations, max_radius)
6:     dataset ← dataset \ data_covered_by(BEST_DETECTOR)
7:     not_covered_points ← size(dataset)
8:     detectors_set ← detectors_set ∪ BEST_DETECTOR
9: end while
```

the detectors are designed to never match any of the strings in S. The drawbacks found in the NS with binary representation have encouraged the development of a Real Valued Negative Selection Algorithm (RNS) [16], which has the same three basic steps of the original NS, but employs real-valued encoding for the characterization of self and nonself. The detectors and self points can be seen as hyperspheres; the self points represent normal behaviour and the detectors are responsible for finding anomalies. The detectors radius determines the threshold for the matching rule, which in this case is the Euclidean distance. This algorithm takes the self samples, which are represented by n-dimensional points, and tries to evolve detectors (another set of n-dimensional points) to cover the nonself space. This is performed through an interactive process aiming at two main goals: keep the detectors away from the self set and maximise the nonself space coverage by keeping the detectors apart. The hyperspace covered by each detector can be set through a parameter r (radius). Once detector generation is completed, the detectors can be employed in the third phase (monitoring).

In the original RNS, in which the number of detectors to be used has to be chosen beforehand, there is no way to determine if the algorithm is really improving the placement of the detectors in order to provide the best possible distribution. In a subsequent version of the algorithm [17], the number of detectors can be estimated beforehand and their positions are determined by the minimization of a function that represents the overlap between the detectors themselves and between them and the points that represent the self. In these implementations of RNS, all the detectors have the same radius, which may pose a scalability limitation: if a small radius is chosen, a large number of detectors will be needed to cover the space, while if a large radius is chosen, parts of the nonself space may be left uncovered.

One of the most critical aspects related to the negative selection algorithm regards the presence of holes in the non-self space coverage. Such holes correspond to undetectable nonself regions that did not match with the detectors. The number of holes is proportional to the radius r of the detectors and to the similarities in the self set (the structure of the self set). Indeed, as r decreases, the number of holes increases. Variants to the RNS algorithm are developed in order to come up with a set of detectors that cover the complement space to the normal one, in order to classify new, unseen data as normal or anomalous. The use of variable shape and size of detectors helps to better cover holes and to address the high dimension problems in such real-valued spaces. For example, Dasgupta and colleagues [4] considered variable shapes such as hyperrectangles, hyperspheres, and hyperellipses, for covering the non-self space, while other works [20, 34] presented a discussion concerning the usefulness and the fairness of the negative selection technique for anomaly detection.

Nevertheless, for simplicity, at the first step, the approach implemented in this work considers the hyperspherical detector model, in which each normal sample is defined as a hypersphere centered in c_i with constant radius r_s, i.e. $s_i = (c_i, r_i)$, $i = 1, \ldots, l$, where l is the number of input samples. Also the detectors d are represented as hyperspheres: $d_j = (c_j, r_j)$, with $j = 1, \ldots, p$, where p is the number of detectors generated, with center c_j and radius r_j. In the first step, the radius of detectors is randomly generated within a given range, and the detector set is generated by checking whether each hypersphere generated contains at least one normal sample: only the detectors that do not include them are kept; the others are discarded. In this algorithm the radius of a new detector is set to the Euclidean distance of the nearest normal case from its center c_j in the range $[0, \texttt{MAX\_RADIUS}]$. The general steps of the negative selection algorithm are reported in Algorithm 3.2 and its parameters are shown in Table 3.6.

Algorithm 3.2 Negative Selection Algorithm for Anomalous Space Covering

```
dataset, NDETS, dets_options ← load_dataset(dataset, NDETS, options)
detectors_set ← ∅
while number of detectors < NDETS do
    det(c,r) ← create_detector(dets_options)
    if Euclidean_distance(det, dataset) < det_radius then
        discard_detector()
    else
        detector_set ← add(det)
    end if
end while
anomalous_check_detection(detector_set,dataset)
```

3.3.3 Machine Learning Approaches

As previously indicated by Glickman and colleagues in [15], within the domain of machine learning, the learning phase of a binary classification task typically falls under the heading of supervised learning, where the goal is to learn a mapping from inputs to outputs. In such a system each training example would pair a specific input vector, containing the values of the technical indicators, with its associated target output value or label, i.e., normal or anomalous. After training, the system would be presented with new inputs (which might or might not have been observed during training) and asked to infer the target output values.

A large number of machine learning algorithms were developed and applied effectively on a wide range of problems in the last decades, and a complete introduction can be found in the literature [39]. In this work we considered the WEKA data mining software platform [38] for the application of a set of stable and well-known machine learning algorithms. Table 3.2 shows the list of such algorithms and it briefly indicates the main information and the values that the corresponding parameters assumed during the experiments. Detailed information about each technique are reported in the references. It is worth noting that neural network algorithms (like `MLP` and `RBF`) are to be considered nature-inspired due to their analogy with the brain neuronal network, but in this work we will consider them at first Machine Learning algorithms due to their extensive and well-established use in this field in recent decades.

Table 3.2 Machine Learning algorithms

Algorithm	Parameters settings	Ref.
Radial Basis Function (`RBF`) Neural Network with k-means clustering	10 basis functions, 0.25 minimum standard deviation	[5]
Multilayer Perceptron (`MLP`) Neural Network with back-propagation	Learning Rate 0.3, Momentum 0.2, 500 epochs of training without early stopping criteria	[5]
K*	Global Blending 75	[8]
Nearest Neighbour (`NN`)	-	[1]
RIPPERk (`JRip`)	3 folds	[9]
Alternating Decision Tree (`ADTree`)	5 boosting iterations	[14]
C4.5	3 folds	[29]
Fast Decision Tree (`REPTree`)	30 folds, 2 minimum instances per leaf	[31]

3.4 Feature Selection

Several works presented in the literature define feature selection (also known as subset selection) as a process commonly used in machine learning, wherein a subset of the features available from the data is selected for application of a learning algorithm. The best subset contains the least number of dimensions that most contribute to accuracy. Then we are able to discard the remaining, negligible dimensions. A detailed review concerning the feature selection problem is reported in Sewell [32]. Feature reduction is an important stage of preprocessing. Indeed, by removing most irrelevant and redundant features from the data, it improves the performance of learning models by reducing the effect of the curse of dimensionality (see Bishop [5] and Pudil *et al.* [28]), enhancing generalization capability, and, in simple models, speeding up the learning process. In particular, in this work, such feature selection could become helpful in finding how all the financial technical indicators are correlated and the incidence of their elimination.

Feature selection algorithms typically fall into two categories: feature ranking and subset selection. The former rank the features by a metric and eliminate all features that do not achieve an adequate score. The latter searches the set of possible features for the optimal subset.

A common topic in classification is whether all the data could help to improve the overall performance of the classifier. In fact, machine learning methods are able to 'learn' relationships between the training data provided as inputs and the output of the system (a class in the classification task). In some cases, an input might be irrelevant or redundant or, in the worst case, add noise, thus reducing the performance of the classifier. In this work, we considered the second kind of selection by applying the so-called 'greedy backward elimination' (see Caruana and Freitag [6]), in which, starting will all the variables, at each step, the one that decreases the accuracy the most is removed from the subset.

Optimal feature selection for supervised learning problems requires an exhaustive search of all possible subsets of features of the chosen cardinality (i.e. an exhaustive search), but if large numbers of features are available, this is impractical. For practical supervised learning algorithms, the search is for a satisfactory set of features instead of an optimal set. For this reason we then considered a greedy algorithm for feature reduction. In order to maintain a

reduced computational time we set, for PSO and NS alike, a reduced numbers of detectors equal to, respectively, 500 particles and 750,000 detectors.

3.4.1 Exhaustive Search

As shown in Table 3.1, in our previous work we considered eight different technical indicators as inputs. Given such feature set, its subsets comprising a number of features ranging from 2 to 7 are 246 overall. All such combinations should be examined to find out the optimal subset by exhaustive search. Due to the high computational time needed for such a complete search we decided to concentrate our efforts by only applying an exhaustive search to the subsets with, respectively, seven and six features, carrying out 36 combinations of features.

We denote as $S^{\{N\}}$ the subset of all the inputs but the features contained in set N, e.g., $S^{\{-\}}$ indicates the set of all the features and $S^{\{1,2\}}$ the subset with six features not including the first and second.

The average accuracy that we obtained by applying the exhaustive search is shown, respectively, in Table 3.3 for the subsets of seven, and, for the subsets of six features, in Tables 3.4 and 3.5. The average accuracy is calculated over 10 runs for each parameter setting. The first results obtained from the subsets of seven features show how, with PSO and NS algorithms, the technical indicators of moving average and exponential moving average are less influential than the other input features, while subsets of six features also include the Rate of Change (ROC) technical indicator.

Instead, Figure 3.2 shows the comparison of the average accuracy of 10 simulations with a feature subset of dimension 7 to the accuracy of the simulations with all the 8 features provided as inputs, denoted with accuracy 1.0. This figure gives a simple clue about the importance of each feature on the overall accuracy, as example the first feature (the log return) seems to be critical for the performance of the algorithms, in fact with both approaches the accuracy with the subset $S^{\{1\}}$ is clearly worse than the accuracy with the complete feature sets ($S^{\{-\}}$).

Table 3.3 Comparison among different feature subsets of 7 dimensions for PSO and NS algorithms. Each column refers to the subset where the feature indicated in the head row has been removed. Average accuracy on 10 runs of PSO and NS algorithms after the removal of a feature is provided. The best value for each approach is shown in **bold**.

	r_i	MA50	EMA50	RSI	Mom.	ROC	MACD	SIGNAL
Approach	1	2	3	4	5	6	7	8
PSO	0.611	**0.674**	0.67	0.671	0.65	0.671	0.656	0.659
NS	0.598	0.656	**0.66**	0.645	0.651	0.656	0.64	0.633

3.4.2 Greedy Algorithm

To avoid the combinatorial explosion of an exhaustive search as the number of features grows, a heuristic can be used to cope with the problem. When knowledge of the problem domain is available, a specialized heuristic can be developed. Otherwise, when it is unavailable or too expensive, generic heuristics, like a hill-climbing search may become the best choice [11].

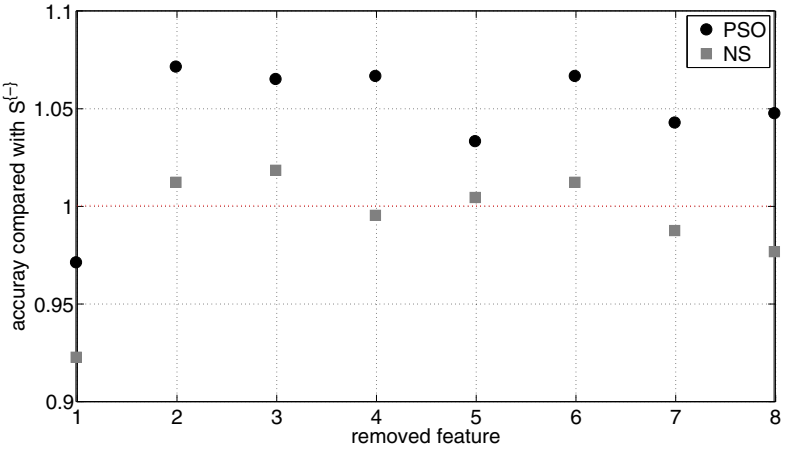


Fig. 3.2 Comparison of the accuracy obtained by using feature subsets of 7 dimensions to the accuracy obtained with $S^{\{-\}}$. On the Y-axis the ratio between accuracy obtained with feature subset and original accuracy is shown

Table 3.4 Comparison of accuracy with different feature subsets of dimension 6 for PSO. Each cell refers to the subset where the features indicated in the relevant column head and row head have been removed. Average accuracy on 10 runs with a subset obtained removing the features indicated by row and column is provided. The best accuracy for each row is shown in **bold**.

	1	2	3	4	5	6	7	8
1	-	0.622	0.608	0.593	0.619	**0.623**	0.554	0.58
2	0.622	-	0.677	**0.687**	0.672	0.683	0.663	0.66
3	0.608	0.677	-	0.684	0.67	**0.685**	0.664	0.66
4	0.593	**0.687**	0.684	-	0.666	0.614	0.647	0.641
5	0.619	**0.672**	0.67	0.666	-	0.667	0.654	0.651
6	0.623	0.683	**0.685**	0.614	0.667	-	0.67	0.67
7	0.554	0.663	0.664	0.647	0.654	**0.67**	-	0.583
8	0.58	0.66	0.66	0.641	0.651	**0.67**	0.583	-

We developed a greedy hill-climbing algorithm, which removes one feature at each step until no improvement is obtained. At each stage, the algorithm evaluates the effects on error criteria of removing one of the m remaining features and then the one with the largest improvement is permanently removed.

In Figure 3.3, a tree with all the features removed is shown for the PSO algorithm. At each iteration, the values reported in brackets correspond to the features discarded from the starting input dataset.

We can see that removing up to 2 features provides better performances than the use of all the features. The features whose removal boosts the performance the most are the RSI index

Table 3.5 Comparison of accuracy with different feature subsets of dimension 6 for NS. Each cell refers to the subset where the features indicated in the relevant column head and row head have been removed. Average accuracy on 10 runs with a subset obtained removing the features indicated by row and column is provided. The best accuracy for each row is shown in **bold**.

	1	2	3	4	5	6	7	8
1	-	0.609	0.605	0.594	**0.65**	0.619	0.595	0.597
2	0.609	-	**0.673**	0.651	0.649	0.661	0.652	0.651
3	0.605	**0.673**	-	0.653	0.66	0.665	0.644	0.656
4	0.594	0.651	**0.653**	-	0.651	0.626	0.639	0.642
5	0.65	0.649	0.66	0.651	-	**0.663**	0.657	0.651
6	0.619	0.661	**0.665**	0.626	0.663	-	0.66	0.657
7	0.595	0.652	0.644	0.639	0.657	**0.66**	-	0.622
8	0.597	0.651	0.656	0.642	0.651	**0.657**	0.622	-

(4) and the 50-days moving average (2), as already made evident by Table 3.4. No further improvement can be obtained by removing a third feature.

Figure 3.4 shows a similar tree with all the features removed for the NS algorithm. In this case, the performance of NS keeps on increasing as more features are removed, reaching an optimum after removal of the 50-days moving average (2), the 50-days exponential moving average (3), momentum (5), and rate of change (6). Surprisingly, the accuracy obtained with such reduced feature set overtakes the best accuracy obtained by PSO, although the latter technique yields generally better results than NS on larger feature subsets. Such results are validated by the implemented exhaustive search, as already reported in Tables 3.4 and 3.5.

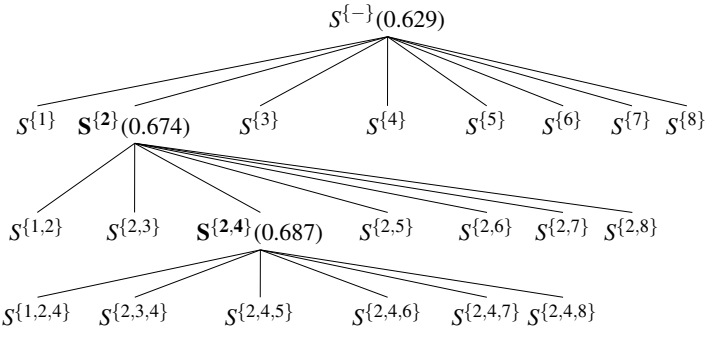


Fig. 3.3 Greedy Algorithm applied on feature selection with PSO algorithm (in brackets the average accuracy on 10 simulations). The subset chosen for each step of the algorithm is denoted in **bold**.

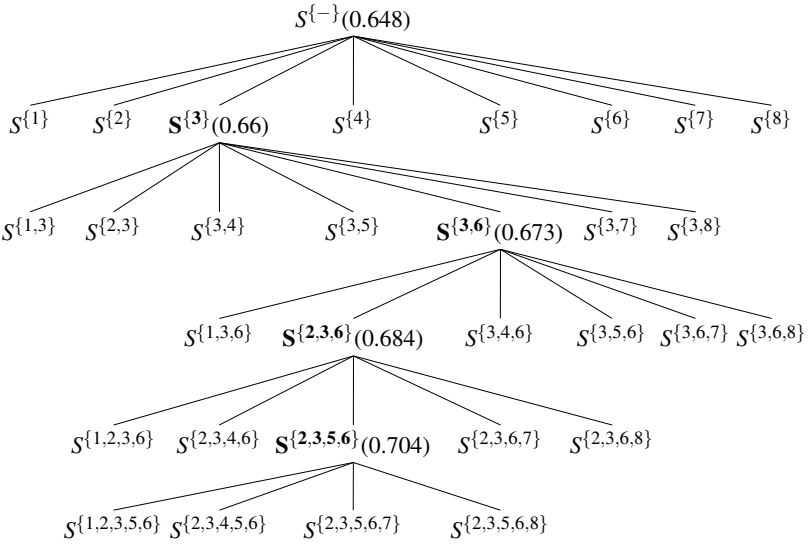


Fig. 3.4 Greedy Algorithm applied on feature selection with NS algorithm (in brackets the average accuracy on 10 simulations). The subset chosen for each step of the algorithm is denoted in **bold**.

3.5 Experiments and Results

In our experiments we have considered a time series of daily prices of the S&P 500 index from April the 14th, 1992 until September the 30th, 2009. The graphical representation of the time series is depicted in Figure 3.5. As previously stated, the considered time series has been divided into two parts in order to define the training and the test sets. In particular, for PSO and NS algorithms, a selection of 2000 normal cases (i.e. continuing a trend) will be chosen from the first part (the dashed shape) in order to create the training set, while the most recent 400 cases (containing both normal and anomalous behaviours in the same quantity) will be used in the test set to evaluate the generalization capabilities on unknown data. Instead, for the machine learning algorithms that we considered for the comparison, we created a different training set, by choosing a set of 2000 normal cases and 2000 anomalous cases (i.e. reversing the trend) from the first dashed shape.

3.5.1 PSO and NS

In this work a first group of experiments is carried out in order to find the optimal settings of the parameters for both PSO and NS algorithms. We performed 10 runs for each setting defined in the two approaches and the results are listed in Table 3.6. Generally, for each of the two algorithms, we noticed that hyperspheres with too small a radius will not be able to cover the anomalies in the space, but, on the other hand, a large radius could have disruptive effects, since the hyperspheres will also cover normal cases, reducing the overall accuracy. The latter represents a critical aspect for the maximum radius. For example, in NS, a maximum radius set to the value 3 was too small, while maximum radius equal to 10 was not

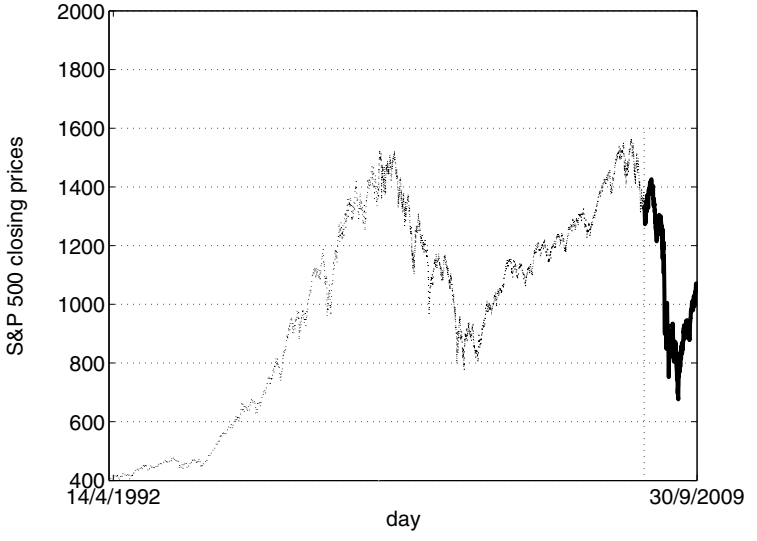


Fig. 3.5 S&P 500 time series. A selection of 2000 normal cases in the fine part of the shape will define the training set. The thick black part of the shape (the last 400 cases) corresponds to the test set.

Table 3.6 Algorithm parameters

Approach	Parameter Description	Value
PSO	Feature Space	$-5,5$
	Maximum Velocity of the particle	2
	c_1, c_2	1.49
	w	[0.4, 0.95]
	Hypersphere Minimum Radius	10^{-4}
	Hypersphere Maximum Radius	1
NS	Feature Space	$-5,5$
	Hypersphere Minimum Radius	10^{-4}
	Hypersphere Maximum Radius	5

enough to improve the performances obtained with values set to 5, with high computational effort. Similar considerations have been carried out about the space in which the features are defined. The experiments show that both algorithms perform well on a solution space defined in the range $[-5,5]$.

At the end of each simulation, calculation of accuracy is performed on the output vector in order to compare the performances for both methods. Accuracy is a relatively straight metric, as reported in Equation 3.5,

$$\text{accuracy}(S) = \frac{\text{TP}+\text{TN}}{\text{P}+\text{N}}, \tag{3.5}$$

where TP and TN are, respectively, the number of anomalous points detected as anomalous and the number of normal points correctly detected. P and N refer to the total number of anomalous (positive) and normal (negative) points.

Since S values of the accuracy, also reported in Table 3.7, are obtained from a balanced dataset of normal and anomalous behaviours, no further normalization is needed for TP and TN values. The performance of the two approaches also depends on the computational effort, represented in this work by calculating the number of evaluations of the distance function in the algorithms. Such a function has been chosen since it represents a fundamental aspect for the fitness evaluation of the PSO algorithm and for the detector generation in the NS algorithm.

The NS algorithm calls the distance function for each point within the training set every time a new detector is introduced, see line 5 of Algorithm 3.2, for this reason the function is called at least a number equals to the dataset length (2000 in our application) times the target number of detectors. For the PSO an estimation of the number of fitness evaluation is harder to obtain due to the complexity of the algorithm, each execution of line 5 of Algorithm 3.1 calls the distance function for a value equals to the dataset length times the product between the particles number and the number of iterations, but the number of points covered by the best detector can differ at each algorithm run.

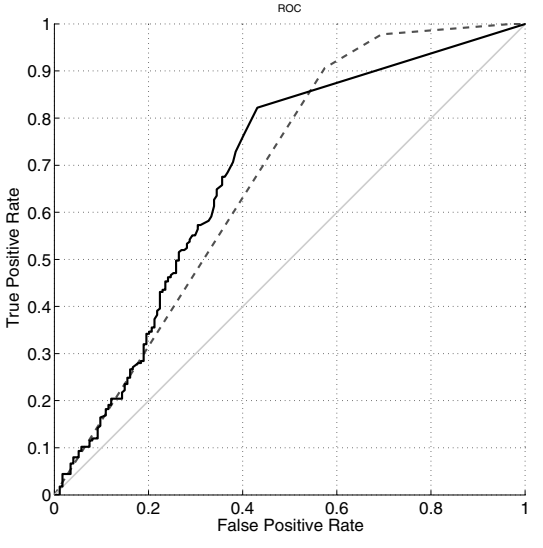


Fig. 3.6 ROC curve of two of the best solutions of both algorithms calculated over the test set. The thick black line is the NS algorithm with 10 millions of detectors, the black dashed line is the best solution of a PSO with 1000 individuals and 300 iterations.

3.5.2 Machine Learning

The anomaly detection approach has been compared with machine learning methodologies, but in order to do that we needed to 'transform' the problem into a binary classification one with:

$$\mathscr{C} = \{C_{\text{anomaly}}, C_{\text{normal}}\} \tag{3.6}$$

Table 3.7 Average results obtained by the two algorithms

Description	Accuracy	TPR	TNR	FPR	FNR	Dist. f. evals.
PSO						
100 ind., 200 iter.	0.60 ± 0.01	0.92	0.21	0.62	0.11	$1.23 \cdot 10^{10}$
100 ind., 500 iter.	0.64 ± 0.02	0.92	0.28	0.57	0.09	$2.49 \cdot 10^{10}$
500 ind., 100 iter.	0.64 ± 0.01	0.81	0.30	0.55	0.11	$2.30 \cdot 10^{10}$
1000 ind., 300 iter.	0.67 ± 0.02	0.89	0.39	0.48	0.16	$7.78 \cdot 10^{10}$
NS						
500,000 ind.	0.63 ± 0.01	0.60	0.67	0.25	0.51	$1.20 \cdot 10^{9}$
750,000 ind	0.64 ± 0.01	0.64	0.64	0.27	0.46	$1.80 \cdot 10^{9}$
1,000,000 ind.	0.65 ± 0.02	0.66	0.64	0.28	0.44	$2.40 \cdot 10^{9}$
5,000,000 ind.	0.68 ± 0.01	0.76	0.60	0.32	0.30	$1.20 \cdot 10^{10}$
10,000,000 ind.	0.70 ± 0.01	0.82	0.55	0.34	0.25	$2.40 \cdot 10^{10}$
20,000,000 ind.	0.71 ± 0.01	0.84	0.53	0.36	0.18	$4.80 \cdot 10^{10}$

where C_{normal} and C_{anomaly} represent the two classes from which we choose the data in order to create the two balanced train and test sets.

We performed several experiments on a subset of classification algorithms contained in WEKA (see section 3.3.3) by considering the same testing set used for the previous simulations, the average results are presented in Table 3.8. Standard deviation and average values are shown where the algorithm is dependent from random conditions, e.g., starting weights for artificial neural network.

Table 3.8 Results (testing) with binary classification dataset with WEKA algorithm

Algorithm	Accuracy	TPR	TNR	FPR	FNR
RBF ANN	0.54 ± 0.02	0.59	0.57	0.53	0.41
MLP ANN	0.58 ± 0.02	0.65	0.52	0.48	0.35
K*	0.57	0.62	0.51	0.49	0.38
NN	0.55	0.61	0.48	0.52	0.55
`Jrip`	0.60 ± 0.01	0.68	0.53	0.47	0.32
`ADtree`	0.62	0.65	0.58	0.42	0.35
C4.5	0.61	0.68	0.54	0.46	0.32
`REPtree`	0.63 ± 0.09	0.69	0.54	0.46	0.31

3.5.3 Using Feature Selection

By applying the feature reduction algorithm we expect that the greedy algorithm leads to a selection of an optimal subset of features which are considered the most significant ones for the detection task we tackled in this chapter. Due to heavy computational times we performed

feature selection by considering as first a reduced set of hyperspheres, respectively 750,000 detectors for NS and 500 particles with 100 iterations for PSO, by assuming that the same considerations could be done for more detectors.

In Table 3.9 the accuracy with the optimal subsets is shown for both algorithm and is compared with the one obtained with the original complete set of features. It is evident how the performance obtained with the subsets found by the greedy algorithm shows a gain, more marked with 'lighter' test configurations. Thus, the feature selection allows a drastic reduction of the computational load on equal accuracy achieved, e.g., the NS with optimal subset achieves the accuracy of 0.704 ± 0.01 with $1.3 \cdot 10^9$ function evaluations while the NS with the complete inputs set to achieve the same accuracy needs a number of evaluations 20 times larger ($2.6 \cdot 10^{10}$).

It should be noted that, after the application of feature selection, the performance difference between PSO and NS has been reduced, as well as the gap among the results of the different algorithm parameter settings, indeed the range of accuracy of tested PSO moved from $0.60 - 0.67$ to $0.68 - 0.695$ and for NS from $0.63 - 0.71$ to $0.696 - 0.722$.

Table 3.9 Comparison of PSO and NS algorithm using as inputs the subset of features obtained by the feature selection

Description	Accuracy	Accuracy with $S^{\{-\}}$	Variation (%)	Dist. f. evals.
PSO $S^{\{2,4\}}$				
100 ind., 200 iter.	0.68 ± 0.01	0.60 ± 0.01	$+13.3\%$	$4.46 \cdot 10^9$
100 ind., 500 iter.	0.687 ± 0.01	0.64 ± 0.02	$+7.3\%$	$1.8 \cdot 10^{10}$
500 ind., 100 iter.	0.681 ± 0.01	0.64 ± 0.01	$+6.4\%$	$1.05 \cdot 10^{10}$
1000 ind., 300 iter.	0.695 ± 0.01	0.67 ± 0.02	$+3.4\%$	$6.24 \cdot 10^{10}$
NS $S^{\{2,3,5,6\}}$				
500,000 ind.	0.704 ± 0.01	0.63 ± 0.01	$+11.7\%$	$1.3 \cdot 10^9$
750,000 ind	0.696 ± 0.02	0.64 ± 0.01	$+8.7\%$	$2.6 \cdot 10^9$
1,000,000 ind.	0.707 ± 0.01	0.65 ± 0.02	$+8.7\%$	$2.6 \cdot 10^9$
5,000,000 ind.	0.722 ± 0.01	0.68 ± 0.01	$+6.1\%$	$1.3 \cdot 10^{10}$
10,000,000 ind.	0.72 ± 0.01	0.70 ± 0.01	$+2.8\%$	$2.6 \cdot 10^{10}$
20,000,000 ind.	0.715 ± 0.005	0.71 ± 0.01	$+0.7\%$	$5.2 \cdot 10^{10}$

3.6 Discussion

As we expected, performances of the two algorithms are quite different. Starting from the first group of experiments particular attention has to be given to the performances with respect to number of distance function evaluations. In Figure 3.6, for example, the best NS solution is obtained with a number of distance function evaluations equal to $4.80 \cdot 10^{10}$, while the best PSO one is obtained with $7.78 \cdot 10^{10}$ evaluations.

The simplicity of NS is an important feature. Indeed, it could be considered as a 'brute force' approach: it creates a large number of simple detectors only by selecting those that do not include the points of the training dataset. Instead, PSO, like other bio-inspired search heuristics, tries to optimize a set of initial points with respect to a particular fitness function.

One of the most critical aspects about the PSO regards the tuning of all the parameters defined in the algorithms, in particular due to the fact that often there are no empirical rules

helping in this choice but only trial-and-error approaches. Moreover, as indicated by Middlemiss, positive selection approaches mainly follow two hypothesis, that are, respectively, the 'Avidity hypothesis' and the 'Differential signaling hypothesis', described in detail in [24]. The main difference between these two ideas of positive selection is that, considering the first hypothesis, the choice of positive or negative selection is dependent on the self/non-self distribution of the problem domain. As opposite, the alternative view of the differential signalling hypothesis (DSH) assumes that the interactions required for positive and negative selection are different. Nevertheless, the motivating idea behind these techniques is to reduce the false positive rates in anomaly detection problem domains.

On the other hand, NS could require a huge amount of detectors in order to cover the space, increasing the computational effort of the approach. Recently, Stibor and colleagues [34] presented a discussion concerning the usefulness and the fairness of the negative selection technique for hamming and real-valued shape-spaces problems in the anomaly detection. Together with finding an optimal distribution of the detectors, i.e. the minimum number of detectors covering the maximum possible non-self space, one of the most important aspects reviewed by the authors and that has to be considered in a real-valued negative selection approach is the self-radius r_s. Indeed, it is the main parameter which influences the classification performance and enables the learning capability (generalization). According to the authors [34], the main problem concerns the value definition, when a training set which contains only self elements is considered and when no techniques (probabilistic or deterministic) are defined to obtain information about the other class. Moreover, the radius r_s strongly depends on the probability density function of the data set, which is unknown a-priori; indeed, an improper chosen radius consequently results in a weak classification performance.

In this chapter, the accuracy shown in Table 3.7, summarizes the computational effort of the two implemented algorithms, proportional to the increasing amount of detectors needed for covering the overall space. However, the small and quite similar values obtained for the standard deviations of accuracy rates for all the experiments carried out allow us to define these as two stable methods.

We can also see that, by doubling the number of distance fitness evaluations, the performances increase about $7-8\%$ in the PSO algorithm, while with negative selection we observe smaller increments. In particular, the accuracy increases of about 3% from 500,000 to 1 million and from 5 to 10 millions of detectors; while only a rise of 1.5% is shown when the detectors increase from 10 to 20 millions. Such an aspect can be due to the fact that, in all the experiments carried out by considering different parameter settings, NS starts with higher values for accuracy. Therefore, their increment can be slow with respect to the PSO algorithm. NS has been demonstrated to be the most useful approach in the improvement of the accuracy only if the augmenting fitness distance evaluations can be considered less important respect to the overall performances.

Nevertheless, by comparing the false positive and the false negative errors of NS and PSO, we notice that they occur in different areas of the space. This aspect could suggest us that a possible combination of both algorithms could exploit this difference to provide a better space coverage, by reducing the values of such errors.

As underlined by the results obtained from all the experiments, the performance of NS and PSO have been improved by the feature selection algorithms applied in this work. In particular it is possible to notice how such a simple algorithm, like a greedy algorithm, may become helpful in the search for the optimal accuracy through the optimal subset of input features. The application of feature selection permitted to exploit the computational power of the two nature-inspired classification algorithms reducing the search space and thus enhancing the accuracy obtained by the same number of distance fitness evaluations. Furthermore, the

improvement obtained by the elimination of features can be explained in the way that some features are either redundant or unnecessary for the anomaly detection problem.

Finally, a first comparison with the machine learning techniques considered in this work shows how nature-based approaches like PSO and NS are well able to find interesting and satisfactory results in this trend-reversal detection problem which are highly competitive with those of other algorithms.

3.7 Conclusion

This chapter investigates the application of two well-known Natural Computing algorithms, the Negative Selection and the Particle Swarm Optimisation, for the task of detecting 'trend reversal points' in the financial time series of the S&P 500 index. Both algorithms are compared with some state-of-the-art machine learning methodologies in order to achieve a better understanding of advantages and drawbacks of hyperspherical detectors based algorithms.

In order to improve the performances of the implemented approaches, they are supplemented with a feature selection algorithm, in order to find high-quality subsets of features for this problem. Experiments carried out have demonstrated how a greedy algorithm may become helpful in this direction, by improving the overall accuracy with a reduced size input features dataset.

As already stated in [3], the NS algorithm produced a better performance than the PSO algorithm, especially on the complete input set, but both outperform binary classification methods, despite the fact that we provided more data in order to define the 'anomalous' class, which is not defined in NS and PSO algorithms.

References

1. Aha, D.W., Kibler, D., Albert, M.K.: Instance-based learning algorithms. In: Machine Learning, pp. 37–66 (1991)
2. Azzini, A., De Felice, M., Meloni, S., Tettamanzi, A.G.B.: Soft computing techniques for internet backbone traffic anomaly detection. In: Giacobini, M., Brabazon, A., Cagnoni, S., Di Caro, G.A., Ekárt, A., Esparcia-Alcázar, A.I., Farooq, M., Fink, A., Machado, P. (eds.) EvoWorkshops 2009. LNCS, vol. 5484, pp. 99–104. Springer, Heidelberg (2009)
3. Azzini, A., De Felice, M., Tettamanzi, A.G.B.: A study of nature-inspired methods for financial trend reversal detection. In: Di Chio, C., Brabazon, A., Di Caro, G.A., Ebner, M., Farooq, M., Fink, A., Grahl, J., Greenfield, G., Machado, P., O'Neill, M., Tarantino, E., Urquhart, N. (eds.) EvoApplications 2010. LNCS, vol. 6025, pp. 161–170. Springer, Heidelberg (2010)
4. Balachandran, S., Dasgupta, D., Nino, F., Garrett, D.: A framework for evolving multi-shaped detectors in negative selection. In: Proceedings of the IEEE Symposium on Foundations of Computational Intelligence, FOCI 2007, pp. 401–408 (2007)
5. Bishop, C.M.: Neural Networks for Pattern Recognition, 1st edn. Oxford University Press, Oxford (1996), http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/0198538642
6. Caruana, R., Freitag, D.: Greedy attribute selection. In: Proceedings of the Eleventh International Conference on Machine Learning, pp. 28–36. Morgan Kaufmann, San Francisco (1994)

7. Chang, J.F., Hsu, S.W.: The construction of stock's portfolios by using particle swarm optimization. In: ICICIC 2007: Proceedings of the Second International Conference on Innovative Computing, Information and Control, p. 390. IEEE Computer Society, Washington, DC, USA (2007), http://dx.doi.org/10.1109/ICICIC.2007.568
8. Cleary, J.G., Trigg, L.E.: K*: An instance-based learner using an entropic distance measure. In: Proceedings of the 12th International Conference on Machine Learning, pp. 108–114. Morgan Kaufmann, San Francisco (1995)
9. Cohen, W.W.: Fast effective rule induction. In: Proceedings of the Twelfth International Conference on Machine Learning, pp. 115–123. Morgan Kaufmann, San Francisco (1995)
10. Colby, R.W., Meyers, T.A.: The Encyclopedia Of Technical Market Indicators. McGraw-Hill, New York (1988)
11. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2003)
12. Falco, I.D., Cioppa, A.D., Tarantino, E.: Facing classification problems with particle swarm optimization. Applied Soft Computing 7(3), 652–658 (2007), http://www.sciencedirect.com/science/article/B6W86-4J61657-1/2/1789a219e3a464b9c22947693cca47c8, doi:10.1016/j.asoc.2005.09.004
13. Forrest, S., Perelson, A.S., Allen, L., Cherukuri, R.: Self-nonself discrimination in a computer. In: Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy, pp. 202–212. IEEE Computer Society Press, Los Alamitos (1994)
14. Freund, Y.: The alternating decision tree learning algorithm. In: Machine Learning: Proceedings of the Sixteenth International Conference, pp. 124–133. Morgan Kaufmann, San Francisco (1999)
15. Glickman, M., Balthrop, J., Forrest, S.: A machine learning evaluation of an artificial immune system. Journal of Evolutionary Computation 13(2), 179–212 (2005)
16. Gonzalez, F., Dasgupta, D., Kozma, R.: Combining negative selection and classification techniques for anomaly detection. In: Proceedings of the 2002 Congress on Evolutionary Computation, CEC 2002, vol. 1, pp. 705–710 (2002)
17. González, F., Dasgupta, D., Niño, L.: A randomized real-valued negative selection algorithm. In: Timmis, J., Bentley, P.J., Hart, E. (eds.) ICARIS 2003. LNCS, vol. 2787, pp. 261–272. Springer, Heidelberg (2003), http://dx.doi.org/10.1007/978-3-540-45192-1_25
18. Herbst, A.: Analyzing and Forecasting Futures Prices. Wiley, Chichester (1992)
19. Jang, G.S., Lai, F., Jiang, B.W., Chien, L.H.: An intelligent trend prediction and reversal recognition system using dual-module neural networks. In: Proceedings of the First International Conference on Artificial Intelligence on Wall Street, pp. 42–51 (1991), doi:10.1109/AIAWS.1991.236575
20. Ji, Z., Dasgupta, D.: Applicability issues of the real-valued negative selection algorithms. In: Proceedings of Genetic and Evolutionary Computation Conference (GECCO), pp. 111–118. ACM Press, New York (2006)
21. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of the IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948 (1995), doi:10.1109/ICNN.1995.488968
22. Jae Kim, K., Han, I.: Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. Expert Systems with Applications 19(2), 125–132 (2000), http://www.sciencedirect.com/science/article/B6V03-40P8SC5-5/2/b16d2b8805a754de8214b5d60b666481, doi:10.1016/S0957-4174(00)00027-0

23. La, T., de Oliveira, A.: Predicting stock trends through technical analysis and nearest neighbor classification. In: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC 2009), pp. 3094–3099 (2009), doi:10.1109/ICSMC.2009.5345944
24. Middlemiss, M.: Positive and negative selection in a multilayer artificial immune system. Tech. Rep. 2006/03, Department of Information Science, Otago (2006)
25. Murphy, J.: Technical Analysis of the Financial Markets. New York Institute of Finance (1999)
26. Peters, E.: Chaos and Order in the Capital Markets, 2nd edn. Wiley, New York (1996)
27. Poli, R.: Analysis of the publications on the applications of particle swarm optimisation. J. Artif. Evol. App. 2008, 1–10 (2008), http://dx.doi.org/10.1155/2008/685175
28. Pudil, P., Novovicov, J., Kittler, J.: Floating search methods in feature selection. Pattern Recognition Letters 15(11), 1119–1125 (1994), http://www.sciencedirect.com/science/article/B6V15-48MPBV9-7/2/c88897a586e12c1a46510ac12fd6c27d, doi:10.1016/0167-8655(94)90127-9
29. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco (1993)
30. Rennard, J.P.: Handbook of Research on Nature-inspired Computing for Economics and Management. IGI Publishing, Hershey (2006)
31. Safavian, S.R., Landgrebe, D.: A survey of decision tree classifier methodology. IEEE Transactions on Systems, Man and Cybernetics 21(3), 660–674 (1991), http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=97458
32. Sewell, M.: Feature selection (2007)
33. Sousa, T., Silva, A., Neves, A.: Particle swarm based data mining algorithms for classification tasks. Parallel Computing 30(5-6), 767–783 (2004), doi:10.1016/j.parco.2003.12.015
34. Stibor, T., Mohr, P., Timmis, J.: Is negative selection appropriate for anomaly detection? In: Proceedings of the International Conference on Genetic and Evolutionary Computation, GECCO 2005, pp. 321–328 (2005)
35. Tax, D.: One-class classification. PhD thesis, TU Delft, Delft University of Technology (2001)
36. Vanstone, B., Tan, C.: A survey of the application of soft computing to investment and financial trading. In: Australian and New Zeland Intelligent Information Systems Conference, pp. 211–216 (2003)
37. Vince, R.: The Handbook of Portfolio Mathematics: Formulas for optimal allocation & leverage. Wiley, Chichester (2007)
38. Witten, I.H., Frank, E.: Data mining: practical machine learning tools and techniques with java implementations. SIGMOD Rec. 31(1), 76–77 (2002), http://doi.acm.org/10.1145/507338.507355
39. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques, 2nd edn. Kaufmann Series in Data Management Systems. Morgan Kaufmann, San Francisco (2005), http://www.worldcat.org/isbn/0120884070
40. Ye, Q., Liang, B., Li, Y.: Amnestic neural network for classification: application on stock trend prediction. In: Proceedings of International Conference on Services Systems and Services Management, 2005 (ICSSSM 2005), vol. 2, pp. 1031–1034 (2005), doi:10.1109/ICSSSM.2005.1500149

4

A Soft Computing Approach to Enhanced Indexation

Nikos S. Thomaidis

Department of Economics, Aristotle University of Thessaloniki, University Campus, Thessaloniki, Greece
and
Department of Financial & Management Engineering, University of the Aegean, 8 Michalon Str., GR-821 00, Chios, Greece
Tel.: +30-2271-0-35454 (35483); Fax: +30-2271-0-35499
nthomaid@fme.aegean.gr

Summary. We propose an integrated and interactive procedure for designing an enhanced indexation strategy with predetermined investment goals and risk constraints. It is based on a combination of soft computing techniques for dealing with practical and computation aspects of this problem. We deviate from the main trend in enhanced indexation by considering a) restrictions on the total number of tradable assets and b) non-standard investment objectives, focusing e.g. on the probability that the enhanced strategy under-performs the market. Fuzzy set theory is used to handle the subjectivity of investment targets, allowing a smooth variation in the degree of fulfilment with respect to the value of performance indicators. To deal with the inherent complexity of the resulting cardinality-constraint formulations, we apply three nature-inspired optimisation techniques: simulated annealing, genetic algorithms and particle swarm optimisation. Optimal portfolios derived from "soft" optimisers are then benchmarked against the American Dow Jones Industrial Average (DJIA) index and two other simpler heuristics for detecting good asset combinations: a Monte Carlo combinatorial optimisation method and an asset selection technique based on the capitalisation and the beta coefficients of index member stocks.

Keywords: Enhanced indexation, Soft computing, Cardinality constraints, Evolutionary optimisation, Simulated annealing, Genetic algorithms, Particle swarm optimisation, Stochastic convergence.

4.1 Introduction

Soft computing is a term encountered in computer science characterising the use of approximate solutions to computationally "hard" tasks for which an exact solution cannot be derived in polynomial time. Many real-life optimisation problems in finance are considered hard due to high dimensionality, combinatorial explosion or the ruggedness of the optimisation landscape. Traditionally, practitioners tackle these problems using ad hoc techniques, expert rules of thumb or computational heuristics that make use of relaxation and decomposition principles (branch & bound, quadratic line-search, etc). Over the last decades, a number of

A. Brabazon et al. (Eds.): Natural Computing in Comput. Finance, SCI 380, pp. 61–77.
springerlink.com

soft computing techniques, such as fuzzy set theory, simulated annealing, genetic algorithms, ant colonies and particle swarms, have been developed as an alternative means of achieving approximate, yet robust and low-cost, solutions to complex problems that were considered unsolvable in the past.

A characteristic class of nonlinear NP-hard optimisation tasks arises in the context of *enhanced indexation*. This portfolio management strategy attempts to generate excess returns compared to more passively placed index funds. Enhanced indexation to a certain extent resembles passive management, as, in principle, fund managers cannot deviate significantly from commercially available indexes, such as those published by Standard & Poors, Russell, etc. However, they also strive to stray from the underlying benchmark with the objective of achieving a better risk-return trade-off, especially in adverse market conditions.

In its simplest form, an enhanced indexation portfolio can be setup by determining a proper capital allocation among all index member securities. However, this strategy is not advantageous for many reasons. First of all, popular benchmark indexes, such as the S&P 500, the Russell 3000 and the Wilshire 5000, have a large number of constituent stocks, some of which are held in very small quantities. Inevitably, the transaction costs or liquidity concerns associated with buying and selling small amounts of stocks may be considerable and can offset the corresponding benefits from trading. Besides, the enhanced fund manager may wish to eliminate securities likely to reduce performance that would be otherwise included in traditional indexes (e.g. low-growth or low-quality companies) or create value though dynamic trading (e.g. buying undervalued or selling overvalued stocks). Taking into account these implementation difficulties, enhanced index funds typically focus on small and manageable bundles of member stocks that are able to achieve the proclaimed investment goal. Successful asset combinations are typically explored by incorporating constraints in the portfolio selection problem, in the form of an upper limit on the *cardinality* (i.e. the size) of the portfolio or the quality characteristics of the tradable assets.

Recently, many techniques have been proposed for structuring cardinality-constraint portfolios with the aim of reproducing the performance of a benchmark index[1]. Despite the numerous applications of passive portfolio management, enhanced indexation problems - especially those incorporating cardinality constraints - have not as yet attracted equal attention. Beasley *et al.* [2] employ a genetic algorithm-like evolutionary method for detecting subsets of stocks that optimally outperform a market index. They adopt a"quasi-passive" formulation of the index tracking problem by maximising a synthetic objective that is a decreasing function of some measure of tracking error and an increasing function of the average deviations of tracking portfolio from index returns. In this way, they search for admissible portfolio weights that generate positive deviations from the index. Similar enhanced indexation formulations have been studied by Gilli and Këllezi [6], who propose an evolutionary heuristic (threshold accepting) to detect optimum asset combinations and portfolio weights. Thomaidis *et al.* [13] consider the problem of actively reproducing the FTSE/ATHEX index using a subset of its constituent stocks. Optimal capital allocations are determined in a mean-variance setting, where the objective is to maximise mean excess (or risk-penalised) return while placing a limit on the tracking-error standard deviation and/or the total risk of the enhanced portfolio (as measured by the variance of portfolio returns). In this chapter, we expand upon this formulation by structuring enhanced indexation strategies using non-typical objectives for investment performance. These focus, for example, on the frequency with which the enhanced portfolio delivers positive return relatively to the benchmark, thus revealing alternative risk aspects of

[1] Some references are given in following sections; see also [3] for a comprehensive survey.

the investment strategy. Such targets/constraints on portfolio performance can be effectively handled within the framework of *fuzzy mathematical multi-objective programming*.

In determining enhanced allocations, we incorporate additional constraints into the optimisation problem, such as a limit on the maximum number of assets included in the portfolio as well as upper and lower bounds on asset weights. From a computational point of view, the incorporation of these complex, though realistic, constraints poses a challenge to traditional numerical methods. To deal properly with the complexity and the ruggedness of the solution space, we apply three nature-inspired heuristics: simulated annealing, genetic algorithms and particle swarm optimisation. This chapter sheds light on the *stochastic convergence properties* of these computational algorithms in the context of enhanced portfolio selection. Instead of relying on the best-ever-found solution, as is typical in many optimisation exercises, we analyse the dispersion of results by estimating the probability of converging to a solution within the range of the global optimum. A similar approach has been suggested by [7] in other optimisation tasks. We believe that our analysis can help provide valuable information to fellow researchers on how to operate each algorithm optimally and what to expect from each of them.

The rest of the chapter is structured as follows: enhanced index tracking with cardinality constraints is discussed in Section 4.2, while Section 4.3 briefly presents the computational heuristics used to solve the associated optimisation problems. Section 4.4 describes a "fuzzy" approach to enhanced portfolio management incorporating approximate investment targets and portfolio constraints. In Section 4.5 we evaluate the performance of optimal portfolios in terms of actively reproducing the American DJIA index. Section 4.6 summarises the main findings and proposes future research directions.

4.2 Enhanced Indexation with Cardinality Constraints

Assume a fund manager with the aim of beating a benchmark index I using only a subset of its tradable assets. For this purpose, he/she collects a sample of observations, $\{r_{t1}, r_{t2}, \ldots, r_{tN}, r_{tI}\} \in \mathbb{R}^{N+1}$, $t = 1, ..., T$, where r_{ti} is the t-period return on the ith asset and r_{tI} denotes the same period's return on the benchmark. The *tracking error*$(r_{t,TE})$ is the *excess* return gained on the portfolio P at the end of the investment period t, i.e. $r_{t,TE} = r_{tP} - r_{tI}$, where $r_{tP} \equiv \sum_{i=1}^{N} r_{ti} w_i$ and w_i is the percentage of capital invested in asset i. Based on the above definitions, the portfolio selection problem can be stated in its general form as:

$$\underset{\mathbf{w} \in R^N, \mathbf{s} \in \{0,1\}^N}{\text{maximise}} \; f(\mathbf{w}, \mathbf{s}) \tag{4.1a}$$

subject to

$$\sum_{i=1}^{N} w_i = 1 \tag{4.1b}$$

$$w^l \leq w_i \leq w^u, \; i = 1, ..., N \tag{4.1c}$$

$$\sum_{i=1}^{N} s_i \leq K \leq N \tag{4.1d}$$

The enhanced portfolio's objective $f(\mathbf{w}, \mathbf{s})$ is a function of the weights configuration $\mathbf{w} = (w_1, w_2, ..., w_N)'$, a vector of binary variables $\mathbf{s} = (s_1, s_2, ..., s_N)'$ and sample data. Several choices for $f(.,.)$ are discussed in Section 4.4. Each s_i is an indicator function that takes the value 1 if the asset i is included in the portfolio and 0 otherwise. All asset weights sum up

to one, implying that the initial capital is fully invested, and the maximum number of assets allowed in the portfolio does not exceed $K \leq N$, which is the so-called *cardinality constraint*. Depending on the cardinality of the portfolio P, some of the w_i's may be zero; while for non-zero weights the fund manager may impose a lower or upper limit on their admissible values, w^l and w^u, respectively, the so-called *floor* and *ceiling constraint*.

The introduction of cardinality constraints significantly increases the computational effort associated with deriving optimal portfolio allocations. In fact, one ends up with a nonlinear mixed-variable programming problem, which even for small values of N poses a challenge to standard numerical techniques. In this chapter, we adopt a solution strategy that overcomes the difficulties of handling cardinality constraints, by introducing a transformation of the decision variables (see [13, 11] for details). The underlying idea is to solve the problem in an unconstrained solution space with continuous decision variables $\mathbf{x} = (x_1, ..., x_N) \in \mathbb{R}^N$ and use a deterministic function to map the optimal vector $\mathbf{x}^*$ onto a feasible portfolio allocation $\mathbf{w}^* = (w_1^*, w_2^*, ..., w_N^*)$, with at least K out of N zero elements. The afore-mentioned mapping technique overcomes computational problems arising from the incorporation of the binary variables s_i in the portfolio selection formulation. However, the resulting unconstrained problem is typically harder to solve due to the existence of multiple local optima, "flat" plateaus and discontinuities. Since, in such optimisation landscapes, traditional gradient-based techniques are prone to premature convergence, the use of more flexible computational heuristics is very much recommended.

4.3 Soft-Computing Optimisation Algorithms

Three popular soft-computing techniques are employed to solve the optimisation problems arising in enhanced indexation tasks. We first present a trajectory-based strategy, namely *simulated annealing*, and then introduce two evolutionary computational heuristics, *genetic algorithms* and *particle swarm optimisation*. Due to space limitations, we are only confined to a summary presentation of the algorithms; more implementation details are provided in the references given below and in Section 4.5.

Simulated annealing(SA) took its name and inspiration from annealing, a technique used in metallurgy involving heating and controlled cooling of a solid ([10]). The central idea is to start with some arbitrary solution and have it modified by randomly generating a number (or else a *population*) of new solutions in its vicinity. A set of acceptable modifications is then defined based on two criteria. A solution with a lower value for the objective function is always acceptable. Additionally, to overcome the possibility of premature convergence to local optima, the *Metropolis* rule is used, according to which changes that come with impairment are accepted with a decreasing probability. The probability is a function of the difference between the corresponding objective values and a parameter T (the computational metaphor of "temperature") that is gradually decreased with the number of iterations by a *cooling factor*. The new solution is randomly drawn from the set of acceptable ones. This process carries on until a set of stopping criteria are met, determined either by the maximum number of iterations or the number of consecutive iterations without change/improvement for the current solution.

Simulated annealing is essentially a local-search technique, as it seeks to iteratively improve an existing solution. An alternative solution-search strategy would be to force a *parallel exploration* of the solution space by means of several agents that interact with each other. This is the idea employed by a *genetic algorithm* (GA). Inspired by the process of natural selection that drives biological evolution, a genetic algorithm repeatedly modifies a population of solutions until it "evolves" towards a "fit" generation ([8, 12]). Among the numerous

forms in which GAs manifest themselves in the literature, in this chapter we adopt a simple version that uses real-valued encoding for candidate solutions. Based on the members of the current population, the algorithm generates a new *pool* of solutions upon which the new generation is compiled. Individuals of the current population are first scaled by their relative *fitness value* (value of the objective function) and the n_s best-ranking (*elite children*) are automatically transferred to the next generation. Then, a fraction of the population, equal to the *crossover rate* p_c, is randomly selected and used for crossover. This process helps fit individuals to exchange good genes in a bid to produce even fitter individuals. The rest of the individuals are subject to mutation; in this process, individual genes are disrupted (with some fixed probability equal to the mutation rate p_m) resulting in a new individual. Due to their parallel exploration capability, GAs are very effective in solving large-scale, hard optimisation problems. Therefore, they are nowadays considered by many financial academics or practitioners as an essential toolkit for computational finance[2].

Particle swarm optimisation (PSO) is an optimisation algorithm inspired by the social behaviour of bird flocks or fish schools [9]. PSO shares many similarities with evolutionary computation techniques such as GAs. The system is initialised with a population of randomly dispersed solutions in a multidimensional space and searches for optima by updating generations. However, unlike GA, PSO is not equipped with evolutionary operators such as crossover and mutation. Candidate solutions, called *particles*, fly through the problem space by gravitating towards optimal particles. As iterations proceed, each member of the swarm changes its position and velocity towards the *global best* solution (i.e. the best solution found so far) and the *local best* solution, which is the best solution detected by the particle. The procedure carries on until no further improvement to the optimum is made or the maximum number of iterations is reached.

4.4 Fuzzy Portfolio Management

In practice, it is common for investors to set performance aspiration levels as well as constraints on the total risk of the investment strategy. Fund managers then attempt to encode these objectives/constraints into suitably formulated mathematical programming problems and derive optimal capital allocations. Traditional programming techniques typically require precise definition of objectives and an accurate expression of preference towards return and aversion towards risk. However, in the real world, investment strategies are often structured around imprecise statements about the desired return or the risk profile of trading positions:

> "*In the prevailing economic state, I* ***would be more or less*** *satisfied with an annual return of 2% above the market*."

> "*The downside probability should* ***not significantly*** *exceed 10%*."

Such "vague" expressions contain useful information for portfolio managers but cannot be easily packed into standard mathematical programming tools. On the contrary, fuzzy optimisation theory offers a very convenient framework for accommodating approximate linguistic-type information, based on the concepts of *fuzzy goal* and *fuzzy constraint*[3].

To illustrate the idea, let us assume that the investor's expectation about a portfolio performance indicator *PI* is to *significantly exceed* a certain level s_0. This imprecise performance target can be formulated using the *S*-shaped function:

[2] See e.g. [1, 4] for a systematic and comprehensive review of the topic.

[3] See e.g. [5] for applications of fuzzy mathematical programming in portfolio management.

$$f(x;s_0,s_1)=\begin{cases}0, & x\le s_0\\ 2\left(\frac{x-s_0}{s_1-s_0}\right)^2, & s_0<x\le\frac{(s_0+s_1)}{2}\\ 1-2\left(\frac{x-s_1}{s_1-s_0}\right)^2, & \frac{(s_0+s_1)}{2}<x\le s_1\\ 1, & x>s_1\end{cases}$$

where x is a variable and s_0, s_1 are free parameters. Through proper choice of s_0 and s_1, one can locate the extremes of the sloped portion of the curve (see upper plot of Fig. 4.1). In the context of fuzzy portfolio management, the x-axis of the plot represents all possible values for *PI* while the y-axis measures the overall degree of objective fulfilment. Note that the investor effectively rejects solutions for which the value of *PI* falls below s_0, by assigning a degree of satisfaction equal to 0. As *PI* increases, so does the goal fulfilment and the investor is practically indifferent between any solution for which the value of *PI* exceeds s_1 (the upper aspiration level).

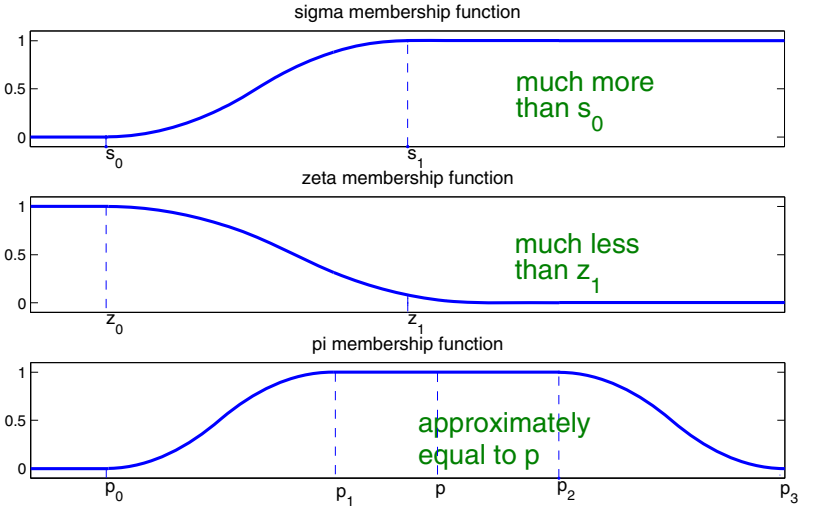


Fig. 4.1 Examples of fuzzy goals and constraints

In a similar fashion, one can formulate goals or constraints of the type "*PI* should be *much less* than z_1" using the Z-shaped function:

$$f(x;z_0,z_1)=\begin{cases}0, & x\le z_0\\ 1-2\left(\frac{x-z_0}{z_1-z_0}\right)^2, & z_0<x\le\frac{(z_0+z_1)}{2}\\ 2\left(\frac{x-z_1}{z_1-z_0}\right)^2, & \frac{(z_0+z_1)}{2}<x\le z_1\\ 1, & x>z_1\end{cases}$$

or of the type "*PI* should be *approximately equal* to p_2" using the Π-shaped function:

$$f(x;p_0,p_1,p_2,p_3)=\begin{cases}0, & x\le p_0\\ 2\left(\frac{x-p_0}{p_1-p_0}\right)^2, & p_0<x\le\frac{(p_0+p_1)}{2}\\ 1-2\left(\frac{x-p_1}{p_1-p_0}\right)^2, & \frac{(p_0+p_1)}{2}<x\le p_1\\ 1, & p_1<x\le p_2\\ 2\left(\frac{x-p_2}{p_3-p_2}\right)^2, & p_2<x\le\frac{(p_2+p_3)}{2}\\ 1-2\left(\frac{x-p_3}{p_3-p_2}\right)^2, & \frac{(p_2+p_3)}{2}<x\le p_3\\ 0, & x>p_3\end{cases}$$

Plots of these functions are given in Fig. 4.1. The fuzzy linguistic framework presented above can be applied in enhanced indexation, by choosing a suitably defined objective function $f(.)$ for the portfolio-selection problem (4.1). In our study, we experimented with two different portfolio objectives, representing realistic enhanced goals:

> Objective 1: *Obtain **some** return in addition to the benchmark, while keeping the total risk of the portfolio **approximately equal** to the benchmark's risk.*
>
> Objective 2: ***Restrict** the probability of under-performing the benchmark, while keeping the tracking error standard deviation **small**.*

These objectives can be formulated by properly combining S-, Z- and Π-shaped functions. In particular, we adopted the following definitions for the maximising function of problem (4.1):

$$f_1 \equiv s(m_{TE},1\%,30\%)\cdot p(s_P,0.99s_I,1.01s_I) \quad \text{and}$$
$$f_2 \equiv z(s_{TE},1\%,10\%)\cdot z(P^-,10\%,50\%) \qquad (4.2)$$

which serve the first and the second portfolio objective, respectively. In the above equations, m_{TE} (s_{TE}) is the average (standard deviation) of tracking error, s_P (s_I) is the standard deviation of portfolio (benchmark) return and P^- is the probability of under-performing the benchmark. A plot of the second objective function is presented in Fig. 4.2. As seen from the particular choice for the cutoff points of f_2, the fund manager is practically unsatisfied with portfolio configurations whose (annualised) s_{TE} is above 10% *or* whose probability of going below the benchmark exceeds 50%. On the contrary, he/she is perfectly happy, and in fact indifferent, between any active formulation that manages to *jointly* keep the tracking error standard deviation below 1% (on an annual basis) and the probability of under-performance below 10%.

4.5 Experiments

4.5.1 Sample Data and Experimental Design

The soft-computing techniques presented above are applied to the task of enhancing the Dow Jones Industrial Average (DJIA) index portfolio. Our sample data includes daily closing values of the DJIA index as well as the prices of its 30 constituent stocks, covering the period

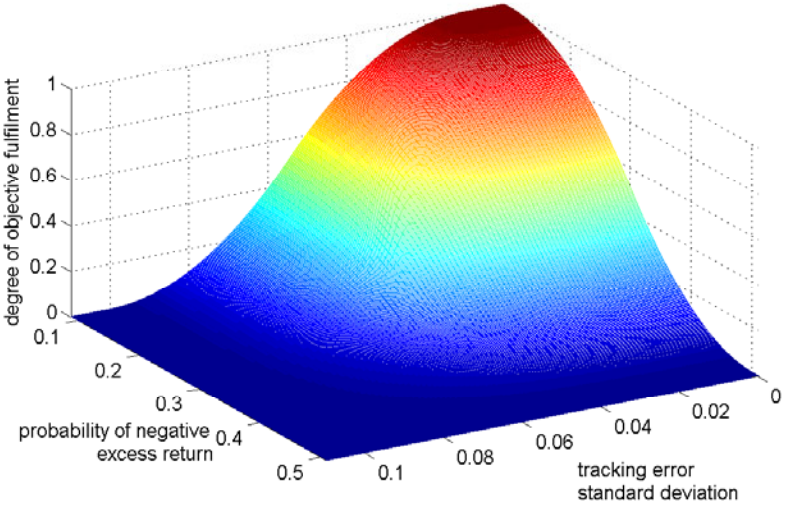


Fig. 4.2 The degree of the investment objective fulfilment as a function of the tracking error standard deviation and the probability of under-performing the benchmark

from 21/01/2004 to 13/01/2006. This amounts to a total of 500 daily return records, half of which are used for deriving optimal portfolio configurations and the remaining half for out-of-sample evaluation of trading performance.

In all optimisation exercises, we assumed that admissible values for portfolio weights lie between 5% and 80%. This way, we avoid investing tiny fractions of capital to DJIA stocks and also prevent single assets from dominating the entire portfolio. We estimated the probability P^- by which a strategy is likely to under-perform the benchmark using the frequency rate T^-/T, where T^- is the number of trading days for which the portfolio's return is below that of the benchmark and T is the total number of observations in the estimation sample. Optimal capital allocations were derived in the estimation sample, by solving the optimisation problem (4.1) for both active objectives described in Section 4.4, and a range of cardinalities $K = \{2,5,10,15,20,25,30\}$. All optimisation algorithms were executed assuming a comparable level of computational resources, using default parameter values suggested in the literature. The population size was set equal to 100 and the maximum number of generations (iterations) was 200. We also performed 500 independent runs of each algorithm from random initial populations. For the simulated annealing process, we set the cooling factor parameter (γ) equal to $0.001^{\frac{1}{200}}$. For the genetic evolution, we encoded individual solutions as real-valued vectors and also adopted a so-called *scattered crossover* scheme, in which parent solutions exchange genes on either sides of a randomly selected coordinate. Mutation was applied by adding to the existing solution a uniformly random vector permutation defined over the feasible range. Of each population, two elite members were duplicated to the next generation and the values for the crossover and mutation rates were set to 0.8 and 0.01, respectively. In the particle swarm

algorithm, the remaining set of parameters were assigned the following values: $w_{min} = 0.001$, $w_{max} = 2$, $c_1 = 2$, $c_2 = 2$.

Nature-inspired optimisation techniques were benchmarked against simpler heuristics for detecting good asset combinations. The first one was a Monte-Carlo portfolio selection technique, in which the optimal portfolio was selected by generating 2000 random asset combinations of fixed cardinality K, computing optimal weights for each combination by means of a gradient-search technique and choosing the asset combination that maximises the portfolio's objective. An alternative stock picking method applied in this study makes more extensive use of market data. All DJIA members are assigned a score based on a linear combination of capitalisation and the index beta coefficient. The portfolio allocation is then computed in three stages by a) sorting all assets in a descending order relative to the portfolio objective, b) selecting members from the top of the list until the cardinality constraint becomes binding (unless the rest of the constraints are not satisfied) and c) computing optimal weights for the particular combination using a gradient-search algorithm. The focus on market cap and beta is motivated by the fact that these are relevant measures to index tracking and enhanced indexation. Small (large) cap stocks have been documented to deviate in terms of risk and return from the market as a whole. The beta coefficient relative to the index is an indication of how closely portfolio returns follow index movements. Hence, stocks with a high beta coefficient can reduce tracking error while stocks will a low beta value are more resistant to index volatility and can help in diversifying away systematic risk.

4.5.2 Computational Results

In Fig. 4.3 we show how the fulfilment of the manager's objective varies with the cardinality constraint. We present the maximum (in-sample) degree of attainment achieved by each optimisation technique in all repetitions. Due to space limitations, we only report results for portfolios maximising the second fuzzy objective (f_2), which is concerned with the tracking error standard deviation and the probability of going below the benchmark. The line segments under the label "Monte Carlo$^{5\%}_{\text{worst}}$", ("Monte Carlo$^{5\%}_{\text{best}}$") represent the 5%-worst (best)-performing combination of assets detected after 2000 Monte Carlo trials[4]. The upper curve, which signifies the frontier of the shaded region, corresponds to the globally optimum portfolio configurations, all indicated by nature-inspired optimisation heuristics[5]. As observed, evolutionary algorithms are by far superior as they manage to deliver much more acceptable portfolios at all cardinalities. On the other hand, optimal allocations based on market capitalisation and beta (see "financial heuristic" curve) were generally unsuccessful in meeting the objectives set by the fund manager. However, for almost all the cardinalities, they were better than the medium-performance portfolios suggested by the Monte Carlo exploration.

Fig. 4.3 designates the importance of carefully exploring the space of feasible asset combinations when actively reproducing a benchmark. A relatively small, yet carefully chosen portfolio can meet the investor's objectives more closely than larger portfolios of arbitrary elements. As seen in Fig. 4.3, with an optimal combination of $K = 5$ DJIA stocks the fund

[4] We also present the median curve, which in Fig. 4.3 falls exactly on the "Monte Carlo$^{5\%}_{\text{worst}}$" line.

[5] In almost all cases, except for very low cardinalities, these are the portfolios selected by PSO. The best portfolios identified by GA or SA typically attained lower values for the objective function, although they were superior to either Monte Carlo or financial stock-picking at all cardinalities. Further details about the relative performance of each method are available upon request form the author.

manager can attain the investment goal to a degree of almost 16%, which is higher than that achieved by any capital allocation indicated by the Monte Carlo confidence curves or the stock-picking technique based on market data. It is more likely that this superiority in performance is due to successful asset combinations and not due to optimal capital allocation, as in all portfolio selection techniques considered in this study some form of weight optimisation takes place.

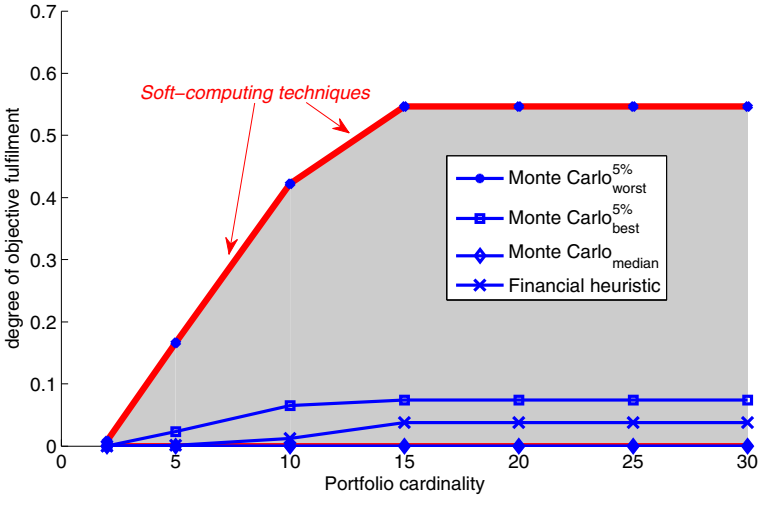


Fig. 4.3 The degree of overall goal attainment vs portfolio cardinality

Fig. 4.4 shows the composition of the globally optimum portfolio found at each cardinality. The position of the squared markers on the vertical axis indicates the index of stocks included in the portfolio and the color of each marker shows the relative importance of each asset in the portfolio. Generally, the darker the color the more weight is assigned to the particular asset. A closer look at what assets are actually selected and what weights they are given reveals another aspect of the complexity associated with choosing enhanced portfolios of limited size. In particular, what makes a good choice in a portfolio with few different assets might not be a good choice for larger portfolios. Hence, the optimal selection with K assets cannot be always determined incrementally by simply augmenting the best solution of $K-1$ assets. This particularly applies to small- or medium-size bundles of stocks, where the "tightness" of cardinality constraint may force the optimisation process to include one asset that serves as a substitute for a bundle of other assets, which cannot be included otherwise. For larger portfolios, the cardinality constraint is not binding as the chosen (floor and ceiling) limits on asset weights become more decisive for the actual synthesis of the optimal enhanced

portfolios. After a certain cardinality, portfolio weights are stabilised and exactly half of the DJIA stocks are assigned zero weights, even though the cardinality constraint gradually becomes looser[6].

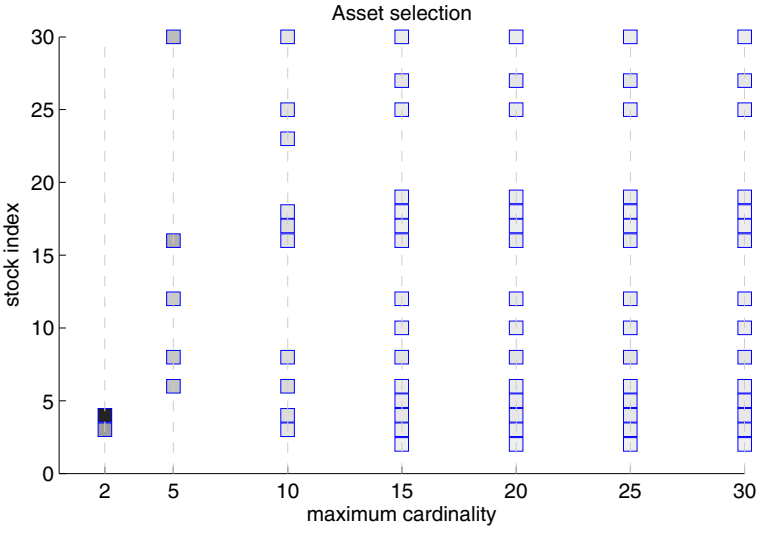


Fig. 4.4 The composition of the globally optimum portfolio at each cardinality

Table 4.1 gives insight into the relative performance of soft-computing optimisation schemes. It reports the percentage of runs for which each algorithm detected a solution that is at worse (10%, 30%, 50%) far from the *global* optimum (the best portfolio found at each cardinality). This serves as an estimate of the probability of reaching an optimal region in a single run. A general remark about the results of Table 4.1 is that the probability of obtaining near-optimum solutions is typically a left-skewed v-shaped function of the portfolio cardinality. This implies that, as we move towards medium-range portfolios, more restarts are needed to detect a good solution with higher confidence. Still, low cardinality problems are typically easier to solve than high cardinality ones. This pattern in the success rate is indicative of two sources of complexity: the combinatorial complexity, which peaks at 15-asset combinations, and the difficulty in deriving optimal weights, which is more apparent in large portfolios. As expected, the success rate quickly builds up at all cardinalities by expanding the definition

[6] As noted by an anonymous referee, the limits on portfolio weights should perhaps vary with the value of K to allow for a more consistent exploration of the solution space. In very small portfolios, the ceiling constraint should become tighter to prevent a single asset from dominating the portfolio. Furthermore, the floor constraint should be perhaps relaxed with an increasing portfolio size, as setting the lower limit too high may render the cardinality constraint not binding beyond a certain cardinality. In this chapter, we followed the common practice in portfolio management to set all constraints of the optimisation problem independently of each other. However, additional experiments performed in this direction suggest that the actual range within which weights take values does not effectively alter the bottom line of our findings.

of the optimal region. Note e.g. that in the case of GA the probability of detecting a solution which at worse deviates 30% from the global optimum ranges between 2.0% and 35.6%. If we increase the tolerance with which the optimum region is defined up to 50%, the average hit rate across all cardinalities climbs up to 78.9%. Among all heuristics considered in this study, PSO seems to be the most reliable optimisation technique, as it manages to detect with higher frequency near-optimum allocations, independently of the initial randomisation of the algorithm. SA has the lowest empirical success rate, which gets particulary worse with an increasing number of trading positions. This signifies the benefits from parallel exploration as opposed to trajectory-search techniques.

Table 4.1 Empirical (%) probability of detecting a portfolio allocation within a range of (10%, 30%, 50%) of the global optimum

Cardinality	Simulated annealing	Genetic algorithm	Particle swarm optimisation
5	(0.8, 12.4, 50.2)	(0.0, 35.6, 86.8)	(5.6, 46.6, 89.0)
10	(0.4, 3.4, 20.8)	(0.0, 10.2, 89.2)	(2.8, 34.0, 88.2)
15	(0.0, 0.6, 14.8)	(0.0, 2.0, 70.4)	(0.8, 11.8, 85.8)
20	(0.0, 0.2, 13.0)	(0.2, 2.6, 70.4)	(1.0, 16.2, 86.4)
25	(0.0, 0.4, 15.2)	(0.0, 3.4, 77.4)	(0.8, 13.8, 84.2)

Table 4.2 Stochastic convergence analysis of optimisation heuristics

Heuristic	**Number of iterations**	**Cardinality (K)**				
		5	**10**	**15**	**20**	**25**
SA	1	∞	∞	∞	∞	∞
	10	∞	∞	∞	∞	∞
	20	498	∞	∞	∞	∞
	50	61	498	1497	1497	∞
	100	31	186	748	1497	748
	150	25	99	498	1497	748
	200	23	87	498	1497	748
GA	1	1497	∞	∞	∞	∞
	10	25	1497	∞	∞	∞
	20	15	249	∞	∞	748
	50	11	74	1497	299	299
	100	9	42	373	165	165
	150	8	32	213	135	114
	200	7	28	149	114	87
PSO	1	∞	∞	∞	∞	∞
	10	44	∞	∞	∞	∞
	20	9	74	1497	∞	748
	50	6	17	114	124	135
	100	6	11	43	31	36
	150	5	8	25	18	23
	200	5	8	24	17	21

Table 4.2 sheds light on the *stochastic convergence* of heuristic optimisation techniques. Fixing the number of iterations (generations) of the algorithm and the cardinality of the portfolio, we report the minimum number of restarts needed to detect (with 95% confidence) a solution that is at worse 30% far from the global optimum. This number is calculated as follows: for each cardinality K and each iteration t ($t = 1,...,200$) we calculate the frequency $\hat{p}(t,K)$ of detecting a solution within the range $[0.7f_2^{min}(K), f_2^{min}(K)]$, where $f_2^{min}(K)$ is the score of the globally best solution found for a portfolio of maximum size K (after 200 iterations). If we interpret this as the probability of having a successful outcome in a single trial, the minimum number of independent restarts needed to detect a near-optimum solution (in the sense described above) can be determined using critical values from the binomial distribution. In particular, we ask for the minimal number of repetitions such that the probability of observing at least one successful event is no less than 95% (see also [7] for a similar approach). Each cell of Table 4.2 reports the number of restarts $\hat{N}(t,k)$ so that the probability that the algorithm detects a near-optimum solution in t iterations is no less than 95%. A "∞" symbol is used whenever the algorithm failed to converge in the designated optimum region, within the number of iterations reported in the second column[7].

A general remark about the results of Table 4.2 is that with an increasing number of iterations, fewer restarts are needed to detect a good solution (the convergence effect). In low-cardinality portfolios, convergence is relatively easy. For example, the particle swarm heuristic is 95% likely to detect an acceptable portfolio of maximum 5 assets in only 10 iterations, after 44 restarts from random initial states. If the algorithm is allowed to iterate more times, less than ten runs may be enough for quasi-optimality to be attained. Table 4.2 offers another view of the relation between problem complexity and cardinality, seen already from the results of Table 4.1. Typically, the computational effort associated with detecting optimal portfolio allocations climbs up towards medium-range cardinalities. An asymmetry with respect to portfolio size is also observed, in the sense that optimisation heuristics converge faster in small rather than large portfolios. Comparing the relative convergence speed, PSO is deemed the most efficient technique as the amount of computational resources (measured by the required number of independent runs) shows the maximum decline rate with the number of iterations. This result is generally in agreement with the findings of Table 4.1 and seems independent of the size of the portfolio.

4.5.3 Financial Implications

Tables 4.3 & 4.4 give us a picture of the financial performance of the globally optimum portfolios found at each cardinality. Each entry in the tables is an average of the corresponding performance indicator over the whole range of cardinalities. All enhanced indexation strategies were evaluated in terms of:

- the standard deviation of the *realised* tracking error (s_{TE}). The realised tracking error is computed as the average difference between the daily ex-post strategy and index returns.
- the percentage of sample days for which the strategy under-performs the benchmark (P^-).
- the average realised return (m_P).

[7] Of course, the infinity symbol should not be interpreted literally in this case. As an anonymous referee pointed out, it does not take an infinite amount of random starts to come within 30% of the optimum, as a Monte Carlo search could perhaps convergence in the optimum region in a finite - although too many - number of repetitions.

- the standard deviation of realised returns (s_P).
- the Sharpe ratio (SR), i.e. the average realised return in excess of the risk-free rate over the ex-post standard deviation of returns.
- the Sortino ratio (SoR), i.e. the average excess return over the downside standard deviation (measured as the average of squared negative portfolio returns).
- the cumulative return generated by the end of the investment period (CR).

The first three columns of Tables 4.3 & 4.4 also report the degree of fulfilment for the composite as well as for the two individual portfolio objectives (Obj1 refers to the constraint on s_{TE} and Obj2 to the restriction on the probability of shortfall). For comparison purposes, we also report, in the last row of each table, performance measures for a buy-and-hold strategy (B&H) allocating equal proportion of capital to all member stocks. The B&H portfolio in this context is not used as an alternative active management strategy but as a reference point for the trading performance of enhanced indexation portfolios. Thus, it allows us to judge how well our proposed strategies performed in the chosen sample period in comparison to the market portfolio. To facilitate interpretation of results, we express figures as a percentage and on an annual basis (except for P^- which refers to the probability of shortfall between two consecutive trading days). All calculations of SR assume a constant risk-free rate of return equal to 3% per annum.

The overall picture of Tables 4.3 & 4.4 indicates both in- and out-of-sample superiority of optimal portfolios derived by evolutionary techniques. Such asset allocations manage to keep the tracking error standard deviation below 10%, the least desirable threshold, and also have a better control on the probability of under-performing the index, which in no case exceeds 50% as required by the fund manager. Given the success in individual goal fulfilment, evolutionary algorithms attain the highest average score for the composite investment objective among all portfolio-selection techniques. Despite the fact that evolutionary portfolios are characterised by low tracking error, they manage to deliver above-market mean and cumulative return with similar-to-market risk, hence the major improvement in Sharpe and Sortino ratios.

Note that although evolutionary trading strategies manage to control the tracking error in both sample data sets (as seen by the s_{TE} column), they generally find it hard to significantly outperform the B&H portfolio in the second period. The probability of going below the market on average increases from 30.82% in the estimation to 46.98% in the out-of-sample period, thus the reduction in the Obj2 and the total degree of fulfilment. The increasing frequency of under-performance is also evident from the realised returns. The average difference between the mean return of the enhanced strategies and that of the DJIA drops from 13.6-2.48=11.12% in the in-sample to 8.58-4.28=4.30% in the out-of-sample period. This gradual deterioration in the investment performance could be attributed to the fact that portfolios are rebalanced only once per year and hence fail to keep track of possible changes in the dependence structure of stock returns or the market conditions in general.

4.6 Discussion – Further Research

This chapter demonstrates how heterogeneous soft-computing techniques can effectively work together in tackling real-world aspects of portfolio management. We consider the investment situation whereby a fund manager structures a portfolio that aims to track a benchmark index but also attempts to boost returns by straying from the index. We deviate from the main trend in enhanced indexation by considering non-standard objectives focusing on the probability that the investment strategy under-performs the market. Fuzzy set theory is used to handle the subjectivity of performance targets, allowing a smooth variation in the degree

Table 4.3 In-sample (%) performance indicators of optimal portfolios

Portfolio selection method	Degree of fulfilment			s_{TE}	P^{-}	m_P	s_P	SR	SoR	CR
	Total	Obj1	Obj2							
Soft-computing techniques	39.75	71.66	48.07	4.20	30.82	13.60	11.11	94.33	150.07	14.63
Financial heuristic	2.37	46.41	3.90	6.17	44.99	11.07	9.92	79.82	120.22	11.76
Monte-Carlo search	11.00	14.74	70.64	4.22	39.41	9.02	11.17	52.68	98.67	9.56
Buy-and-hold	-	-	-	-	-	2.48	10.76	-4.82	-7.58	2.51

Table 4.4 Out-of-sample (%) performance indicators of optimal portfolios

Portfolio selection method	Degree of fulfilment			s_{TE}	P^{-}	m_P	s_P	SR	SoR	CR
	Total	Obj1	Obj2							
Soft-computing techniques	1.28	65.44	1.59	4.61	46.98	8.58	11.09	49.96	86.11	8.99
Financial heuristic	0.01	55.11	0.43	5.33	49.37	2.54	9.96	-4.73	-2.34	2.66
Monte-Carlo search	0.27	67.37	0.35	4.42	49.03	4.52	11.2	14.65	11.56	4.67
Buy-and-hold	-	-	-	-	-	4.28	10.05	12.70	20.89	4.37

of fulfilment with respect to the value of performance indicators. Nature-inspired optimisation heuristics are employed for dealing with the computational complexity of the resulting enhanced indexation formulations, especially after the introduction of cardinality constraints. Optimal portfolios derived from approximate optimisers are then benchmarked against the American Dow Jones Industrial Average (DJIA) index and two other simpler heuristics for detecting good asset combinations: a Monte Carlo-based combinatorial optimisation method and an asset selection technique based on the capitalisation and the beta coefficients of index member stocks.

Our experimental results provide valuable insight into the quantitative and computational aspects of enhanced index investing. We show that the ultimate success of an indexation strategy depends both on careful asset selection and optimal capital allocation. The simple expert rule of thumb based on market information manages to "filter out" assets that ought not to be included in any of the optimal portfolios but still ignores valuable information contained in the dependence structure of stock returns (at least the proportion of the dependence that is not captured by market-wide events). The Monte Carlo approach takes that into account by looking at combinations of assets, but is also likely to pick one or several stocks with undesirable characteristics. For these reasons, evolutionary heuristics that perform an exploration of the solution space in *both* directions are more likely to detect enhanced portfolio allocations with improved in- and out-of-sample performance.

Computation methods, such as genetic algorithms and particle swarms, are shown to be an effective tool for handling the inherent complexities of cardinality-constrained portfolio optimisation problems. These optimisation heuristics combine exploration rules, inspired by physical/biological processes, with random elements in the process of choosing optimal directions in the solution space and/or when deciding whether to replace current solution(s) with new ones. Deliberately introducing randomness into the search process is a good means of avoiding premature convergence and moving towards global optima. It however implies a stochastic approximation to the global optimum, where the convergence path and possibly the reported solutions change in each run of the algorithm. Then comes the issue of how much computational resources have to be spent before a near-optimum solution is reached with high confidence. Since we generally lack universally-applicable theoretical results with regard to the convergence speed of nature-inspired heuristics, this problem has to be addressed on an empirical basis. Following this direction, we perform an empirical analysis of the convergence properties of simulated annealing, genetic algorithms and particle swarms in the context of enhanced indexation. Instead of reporting the best solution found in each run, we derive the *probability distribution* of results by calculating the frequency of reaching a solution in the vicinity of the optimum. We demonstrate how this information could be further exploited in accessing the minimal amount of computational resources (i.e. the number of restarts and the number of iterations per restart) that should be reserved given the complexity of the optimisation task.

The empirical evidence provided in this chapter clearly shows that there is much potential in applying soft-computing techniques in enhanced portfolio management. There are however several implementation issues that have to be addressed before the presented methodology could be deployed on a larger scale. Perhaps, one weak point is the considerable number of free parameters and the lack of clear guidelines on how to choose optimal values. At the current stage of our methodology development, many design parameters, such as the cutoff points in fuzzy goals, were selected based on advice from experts and realistic market aspirations. We did not put considerable effort in deriving optimal parameter values, although preliminary experiments show that the fund manager could obtain allocations with a better trade-off between expected risk and reward by properly adjusting the cutoff points of the

objective functions. Still, setting extreme investment targets for in-sample portfolio performance may lead to "edge" allocations with poor results on unseen data. Despite the fact that many of the problems discussed above are also encountered in "non-fuzzy" formulations of enhanced indexation, it would be wise to perform a sensitivity analysis with respect to these variables. For instance, one could examine how much the degree of objectives fulfilment, the portfolio synthesis and the trading performance of optimal allocations vary with the tolerance to constraint violation, the least acceptable probability of under-performance or the tightness of the cardinality constraint. Furthermore, it would also be interesting to see whether better performance can be achieved by utilising different definitions of portfolio reward and risk. Some benchmarking against other computational heuristics or simple rules of thumb for detecting good asset combinations, would shed further light on the complexities associated with designing enhanced indexation strategies.

References

1. Bauer, R.J.: Genetic Algorithms and Investment Strategies. Wiley, Chichester (1994)
2. Beasley, J.E., Meade, N., Chang, T.J.: An evolutionary heuristic for the index tracking problem. European Journal of Operational Research 148, 621–643 (2003)
3. Canakgoza, N.A., Beasley, J.E.: Mixed-integer programming approaches for index tracking and enhanced indexation. European Journal of Operational Research 196(1), 384–399 (2008)
4. Chen, S.H.: Genetic Algorithms and Genetic Programming in Computational Finance. Kluwer Academic Publishers, Dordrecht (2002)
5. Fang, Y., Lai, K.K., Wang, S.: Fuzzy Portfolio Optimization: Theory and Methods. Lecture Notes in Economics and Mathematical Systems, vol. 609. Springer, Heidelberg (2008)
6. Gilli, M., Këllezi, E.: Threshold accepting for index tracking. working paper (2001)
7. Gilli, M., Winker, P.: Review of Heuristic Optimization Methods in Econometrics. In: COMISEF Working Papers Series WPS-001 (2008)
8. Goldberg, D.E.: Genetic Algorithms in Search. In: Optimization and Machine Learning. Kluwer Academic Publishers, Dordrecht (1989)
9. Kennedy, J., Eberhart, R.C., Shi, Y.: Swarm Intelligence. Morgan Kaufmann Publishers, San Francisco (2001)
10. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 220, 671–680 (1983)
11. Maringer, D., Oyewumi, O.: Index tracking with constrained portfolios. Intelligent Systems in Accounting, Finance and Management 15, 57–71 (2007)
12. Mitchell, M.: An Introduction to Genetic Algorithms. MIT Press, Cambridge (1996)
13. Thomaidis, N.S., Angelidis, T., Vassiliadis, V., Dounias, G.: Active Portfolio Management with Cardinality Constraints: An Application of Particle Swarm Optimization. Special Issue on New Computational Methods for Financial Engineering, Journal of New Mathematical and Natural Computation 5(3) (2009)

5

Parallel Evolutionary Algorithms for Stock Market Trading Rule Selection on Many-Core Graphics Processors

Piotr Lipinski

Institute of Computer Science,
University of Wroclaw, Wroclaw, Poland
lipinski@ii.uni.wroc.pl

Summary. This chapter concerns stock market decision support systems that build trading expertise on the basis of a set of specific trading rules, analysing financial time series of recent stock price quotations, and focusses on the process of rule selection. It proposes an improvement of two popular evolutionary algorithms for rule selection by reinforcing them with two local search operators. The algorithms are also adapted for parallel processing on many-core graphics processors. Using many-core graphics processors enables not only a reduction in the computing time, but also an exhaustive local search, which significantly improves solution quality, without increasing computing time. Experiments carried out on data from the Paris Stock Exchange confirmed that the approach proposed outperforms the classic approach, in terms of the financial relevance of the investment strategies discovered as well as in terms of the computing time.

5.1 Introduction

Many-core graphics processors, which appeared in recent years, provide a new type of efficient parallel computing platforms and create new opportunities for computationally intensive approaches which typically require long run-times. Numerous applications of many-core graphics processors include data clustering [4], neural networks [3] or optimization algorithms [17] and concern many domains, such as astrophysics, bioinformatics, operational research and finance.

Although a special architecture of many-core graphics processors gives an opportunity to split time-consuming computation into a large number of threads and run hundreds of threads simultaneously, it requires a non-standard manner of designing and programming algorithms, and consequently makes the computing environment efficient only for certain types of applications. A special technique of memory management, which causes some restrictions in capacities and bandwidths, may also constitute a bottleneck for certain algorithms, which involve extensive processing of large data volumes, and therefore requires some effort in algorithm engineering.

One of the enticing open challenges concerns the application of many-core graphics processors to computational finance. Growing interest in computational approaches to financial

A. Brabazon et al. (Eds.): Natural Computing in Comput. Finance, SCI 380, pp. 79–92.
springerlink.com

modelling has led to numerous potential applications for many-core graphics processors, including, the application of evolutionary algorithms and neural networks for stock market data analysis [6], [8], [10], the use of genetic programming to building decision trees for supporting financial decision-making [16], and evolutionary approaches for portfolio optimization [11], [12]. Other applications include money management [14] or option pricing [5].

This chapter concerns stock market decision support systems that build trading expertise on the basis of a set of specific trading rules analysing financial time series of recent stock price quotations. One of the most important issues influencing the efficiency of the trading expertise is a proper rule selection process. It is often defined as an optimization problem with an irregular, large, search space rendering it suitable for the application of evolutionary algorithms.Although evolutionary approaches are capable of finding relatively efficient solutions, they are susceptible to overlook some interesting ones and prematurely converge to a local maximum. Reinforcing evolutionary algorithms by an exhaustive local search often leads to a significant improvement in solution qualities, but requires an unacceptable long computing time. However, using many-core graphics processors enables not only a reduction in the computing time, but also an exhaustive local search, which significantly improves solution qualities, without increasing computing time.

This chapter is structured as follows: Section 5.2 defines the optimization problem of rule selection. Section 5.3 describes the evolutionary algorithm based on the Simple Genetic Algorithm to rule selection and Section 5.4 describes another one based on the Population-Based Incremental Learning. Section 5.5 proposes two local search operators to improve the process of rule selection and Section 5.6 proposes their parallelization. Section 5.7 discusses results of a number of experiments performed on real data from the Paris Stock Exchange. Finally, Section 5.8 concludes the chapter.

5.2 Problem Definition

One of the popular approaches in constructing decision support systems for stock market trading is to use a number of trading rules based on Technical Analysis [13], combining them into a 'trading expert' that provide trading decisions.

Such a trading rule may be defined as a function $f : \mathcal{K} \mapsto s \in \mathbb{R}$ that maps factual financial knowledge $\mathcal{K}$ (for example, a financial time series of recent stock price quotations) to a real number s encoding a trading signal: a sell signal for the value of s below a certain threshold θ_1, a buy signal for the value of s above a certain threshold θ_2, and no signal otherwise.

Different trading rules may return different trading signals, may have different efficiency in different stock market conditions, and may have different reliability, so decision support systems often combine single trading rules into sets, called trading experts, and consider their trading signals. Such a trading expert may be defined as a subset E of the entire set $\mathcal{R}$ of some specific trading rules $f_1, f_2, \ldots, f_d$ available in the decision support system, which defines the trading signal of the trading expert, for a given factual financial knowledge $\mathcal{K}$, as the arithmetic average of trading signals of trading rules included in the subset E:

$$\frac{1}{|E|} \sum_{f \in E} f(\mathcal{K}). \tag{5.1}$$

Performance of a trading expert may be defined by a type of simulation over a predefined training period. It starts with an initial capital, an initial amount of cash and an initial number of stocks, at the beginning of the training period. In the successive days of the training period,

the trading expert provides a trading signal. If it is a buy signal, a predefined part of the available cash is invested in stocks. If it is a sell signal, a predefined part of the available stocks is sold. Each transaction is charged with a transaction fee. Finally, the efficiency of the trading expert is defined by the Sharpe ratio [15] of its daily return rates.

Table 5.1 Simulation settings used in experiments

threshold for a sell signal (θ_1)	-0.05
threshold for a buy signal (θ_2)	+0.05
buy limit	50% of cash
sell limit	50% of stocks
initial cash	10000
initial stocks	100
transaction fee	0.39%

Table 5.1 presents the simulation settings applied in experiments. Single trading rules always produce -1 (sell), 0 (do nothing) or 1 (buy) signals. In each transaction either 50% of the available capital was invested in stocks or 50% of the available stocks was sold. The Sharpe ratio of daily return rates was calculated with the daily risk-free return rate of 0.01%.

Therefore, rule selection for constructing efficient trading experts is an optimization problem of finding the trading expert maximizing the objective function being the Sharpe ratio over a given training period. The search space consists of all possible trading experts, being subsets of the entire set of trading rules, so it contains 2^d possible solutions. For a large number of trading rules in the decision support system, the size of the search space constitutes a bottleneck for many optimization algorithms (in the experiments in this chapter, $d = 500$ and $2^d = 2^{500}$).

5.3 Simple Genetic Algorithm for Rule Selection

One of the simplest approaches to solve the optimization problem defined in the previous section may be based on the Simple Genetic Algorithm (SGA) [7], which evolves a population of candidate solutions using standard crossover and mutation operators. Each candidate solution is a binary vector $\mathbf{x} = (x_1, x_2, \ldots, x_d) \in \{0,1\}^d$ representing a trading expert, whose components x_i corresponds to an absence ($x_i = 0$) or a presence ($x_i = 1$) of the i-th trading rule in the trading expert.

Algorithm 5.1 presents the framework of the SGA-RS algorithm. It starts by generating an initial population $\mathscr{P}_0$ composed of N random candidate solutions with a uniform probability distribution. Components of each random candidate solution $\mathbf{x} = (x_1, x_2, \ldots, x_d) \in \{0,1\}^d$ are drawn in such a way that $x_i = 1$ with probability 0.5 and $x_i = 0$ with probability 0.5. After its creation, each candidate solution is evaluated as described in previous sections.

Afterwards, the population evolves until a termination condition is held, usually a predefined number of iterations T. The current population $\mathscr{P}_t$ of size N produces an offspring population $\tilde{\mathscr{P}}_t$ of size $2M$: For each pair of offspring solutions $\tilde{\mathbf{x}}_i$ and $\tilde{\mathbf{y}}_i$, two parent solutions $\mathbf{x}_i$ and $\mathbf{y}_i$ are randomly selected from the current population in such a way that the probability of being chosen for a candidate solution is proportional to its value of the objective function.

Algorithm 5.1 Simple Genetic Algorithm for Rule Selection (SGA-RS)

$\mathscr{P}_0$ = Random-Population();
Population-Evaluation($\mathscr{P}_0$);

$t = 0$;
while not Termination-Condition($\mathscr{P}_t$, t) **do**

  for $i = 1, 2, \ldots, M$ **do**
    $(\mathbf{x}_i, \mathbf{y}_i)$ = Parent-Selection($\mathscr{P}_t$);
    $(\tilde{\mathbf{x}}_i, \tilde{\mathbf{y}}_i)$ = Cross-Over($\mathbf{x}_i$, $\mathbf{y}_i$, p_c);
    Mutation($\tilde{\mathbf{x}}_i$, p_m);
    Mutation($\tilde{\mathbf{y}}_i$, p_m);
  end for

  $\tilde{\mathscr{P}}_t = \{\tilde{\mathbf{x}}_1, \tilde{\mathbf{y}}_1, \tilde{\mathbf{x}}_2, \tilde{\mathbf{y}}_2, \ldots, \tilde{\mathbf{x}}_M, \tilde{\mathbf{y}}_M\}$;
  Population-Evaluation($\tilde{\mathscr{P}}_t$);

  $\mathscr{P}_{t+1}$ = Next-Population-Selection($\mathscr{P}_t \cup \tilde{\mathscr{P}}_t$);

  $t = t + 1$;
end while

Parent solutions produce offspring solutions by one-point crossing over [7] with a cross-over probability $p_c \in [0,1]$ or by simply copying with probability $(1 - p_c)$. Each offspring solution is mutated by negating each component with a mutation probability $p_m \in [0,1]$. Finally, the next population $\mathscr{P}_{t+1}$ is composed from the best candidate solutions from the union of the current and the offspring populations.

Despite its simplicity, the SGA-RS algorithm has some weaknesses, such as a weak resistance for strong individual domination or inefficient random walks in the search space, which often leads to a premature convergence to a local maximum.

5.4 Population-Based Incremental Learning for Rule Selection

Another competitive approach can be based on the Population-Based Incremental Learning (PBIL) [1], one of the simplest Estimation of Distribution Algorithms (EDA) [9], which builds a probability model of efficient candidate solutions. Algorithm 5.2 presents the overview of the PBIL-RS algorithm. It starts by initializing the probability model $\mathbf{p}_0 = (p_1, p_2, \ldots, p_d) \in [0,1]^d$ with $(0.5, 0.5, \ldots, 0.5)$.

Afterwards, the 'evolution' process begins with generating a random population $\mathscr{P}_t$ composed of N random candidate solutions according to the probability model $\mathbf{p}_t$. Components of each random candidate solution $\mathbf{x} = (x_1, x_2, \ldots, x_d) \in \{0,1\}^d$ are drawn in such a way that $x_i = 1$ with probability p_i and $x_i = 0$ with probability $(1 - p_i)$. After creating, each candidate solution is evaluated in a type of a financial simulation as described in previous sections. PBIL-RS takes the best candidate solution $\mathbf{x}^*$ from the current population $\mathscr{P}_t$, updates the probability model $\mathbf{p}_t$ with a learning rate $\alpha \in [0,1]$ and mutates it with a mutation probability $\beta \in [0,1]$ and a mutation rate $\gamma \in [0,1]$. Finally, the evolution process repeats until a

Algorithm 5.2 Population-Based Incremental Learning for Rule Selection (PBIL-RS)

$\mathbf{p}_0 = (0.5, 0.5, \ldots, 0.5)$;

$t = 0$;
while not Termination-Condition($\mathbf{p}_t$, t) **do**

  $\mathscr{P}_t$ = Random-Population($\mathbf{p}_t$);
  Population-Evaluation($\mathscr{P}_t$);

  {updating the probability model};
  $\mathbf{x}^*$ = Find-Best-Solution($\mathscr{P}_t$);
  $\mathbf{p}_{t+1} = (1-\alpha) \cdot \mathbf{p}_t + \alpha \cdot \mathbf{x}^*$;

  {mutating the probability model};
  if random$(0,1) < \beta$ **then**
    u = Random-Binary-Vector();
    $\mathbf{p}_{t+1} = (1-\gamma) \cdot \mathbf{p}_{t+1} + \gamma \cdot \mathbf{u}$;
  end if

  $t = t + 1$;
end while

termination condition is held, usually a predefined number of iterations T. Although the PBIL-RS algorithm is more resistant to strong individual domination, it also often prematurely converges to a local maximum.

5.5 Hybrid Evolutionary Algorithms with Local Search and Simulated Annealing

Although both evolutionary algorithms described in previous sections are capable of finding relatively efficient solutions, they are susceptible to overlook some interesting ones, even if such solutions are close to the candidate solutions from the current population, because the objective function is fairly irregular and a close neighbourhood of a mean candidate solution from the current population may contain a potent candidate solution easy to omit by the standard evolutionary search.

One of the popular techniques to alleviate this issue is to combine the evolutionary mechanism with the additional local search operator that explores a certain neighbourhood of some candidate solutions. Certainly, this may lead to a few problems, such as the Baldwin effect or Lamarckian evolution, but they may be reduced by increasing the population size, decreasing the ratio of additionally optimized candidate solutions or strengthening the mutation operator.

In order to improve the process of rule selection, both evolutionary algorithms described in previous sections may be reinforced by two local search operators: Iterative Local Search (ILS), presented in Algorithm 5.3, and Iterative Local Search with Simulated Annealing (ILS-SA), presented in Algorithm 5.4, which additionally improves either all offspring solutions produced by the crossover and mutation operators or only a part of them chosen according to some selection criteria.

Algorithm 5.3 presents the framework of Iterative Local Search operator (ILS) for improvement of a candidate solution $\mathbf{x}$. It starts by examining all neighbouring solutions of the original solution $\mathbf{x}$, i.e. binary vectors differing from $\mathbf{x}$ on at least one and no more than $\delta_1 \geq 1$ positions, and finding the best neighbouring solution $\hat{\mathbf{x}}$, i.e. the neighbouring solution of the largest value of the objective function. If the best neighbouring solution $\hat{\mathbf{x}}$ outperforms the original solution $\mathbf{x}$, $\hat{\mathbf{x}}$ replaces $\bar{\mathbf{x}}$ and such an optimization process repeats until no improvement is obtained.

Algorithm 5.3 Iterative Local Search (ILS)

{$\mathbf{x}$ is the original candidate solution to optimise}

$\hat{\mathbf{x}}$ = Best-Neighbour-Solution($\mathbf{x}$);
while $F(\hat{\mathbf{x}}) > F(\mathbf{x})$ **do**
  $\mathbf{x} = \hat{\mathbf{x}}$;
  $\hat{\mathbf{x}}$ = Best-Neighbour-Solution($\mathbf{x}$);
end while

Algorithm 5.4 presents an overview of Iterative Local Search with Simulated Annealing operator (ILS-SA) to improve a candidate solution $\mathbf{x}$. It starts with the ILS operator, which transforms the original solution $\mathbf{x}$ into a local optimum $\hat{\mathbf{x}}$, and mutating the local optimum $\hat{\mathbf{x}}$ by negating randomly chosen $\delta_2 \geq 1$ positions. Next, the ILS operator transforms the new solution into a new local optimum $\bar{\mathbf{x}}$. If the new local optimum outperforms the old one, i.e. $F(\bar{\mathbf{x}}) > F(\hat{\mathbf{x}})$, $\bar{\mathbf{x}}$ replaces $\hat{\mathbf{x}}$. Otherwise, with a probability proportional to $\exp(F(\hat{\mathbf{x}}) - F(\bar{\mathbf{x}}))$, the new solution $\bar{\mathbf{x}}$ either remains or is replaced with the local optimum $\hat{\mathbf{x}}$. Finally, the new solution $\bar{\mathbf{x}}$ is again mutated and optimized with the ILS operator. Such an optimization process repeats a predefined number of times.

5.6 Parallel Approach to Rule Selection

Despite improving the process of rule selection, combining evolutionary algorithms with local search significantly increases the computational complexity and consequently makes the approach impractical due to the excessively long computing time. In order to make the approach applicable, rule selection algorithms may be adapted to parallel processing and run on powerful computing platforms such as many-core graphics processors. This chapter refers to many-core graphics processors with the Compute Unified Device Architecture (CUDA), which is a parallel computing architecture for NVidia graphics processors, with a specific parallel programming model and an instruction set architecture.

In the CUDA architecture, the computing platform consists of two parts: a sequential platform with a standard processor and a parallel platform with a many-core graphics processor. The sequential platform executes the single-threaded mainstream of a program and invokes some multi-threaded subprograms on the parallel platform. The parallel platform consists of a number of multi-threaded streaming multiprocessors that execute threads of an invoked subprogram in parallel. Multi-threaded streaming multiprocessors operate in the Single-Instruction Multiple-Thread (SIMT) architecture that allows a further parallelization by running a number of threads concurrently on the same multiprocessor.

Algorithm 5.4 Iterative Local Search with Simulated Annealing (ILS-SA)

{$\mathbf{x}$ is the original candidate solution to optimise}

$\hat{\mathbf{x}}$ = Iterative-Local-Search($\mathbf{x}$);
$\check{\mathbf{x}}$ = Mutation($\hat{\mathbf{x}}$);
$\bar{\mathbf{x}}$ = Iterative-Local-Search($\check{\mathbf{x}}$);
while $t < T$ **do**
  if $F(\bar{\mathbf{x}}) > F(\hat{\mathbf{x}})$ **then**
    $\hat{\mathbf{x}} = \bar{\mathbf{x}}$;
  else
    if random() $> \exp(\frac{F(\hat{\mathbf{x}})-F(\bar{\mathbf{x}})}{t/T})$ **then**
      $\bar{\mathbf{x}} = \hat{\mathbf{x}}$;
    end if
  end if
  $\check{\mathbf{x}}$ = Mutation($\bar{\mathbf{x}}$);
  $\bar{\mathbf{x}}$ = Iterative-Local-Search($\check{\mathbf{x}}$);

  $t = t + 1$;
end while
$\mathbf{x} = \hat{\mathbf{x}}$;

When invoking a multi-threaded subprogram, the computing platform splits its threads into a number of blocks and assigns them to multiprocessors, so that each multiprocessor has one or more blocks of threads to execute. Multiprocessors divide threads from each block into so-called warps of 32 threads and processes them consecutively. Multiprocessors start each thread from the same warp at the same time, but may further desynchronise them due to some conditional instructions, and finish the execution of a warp when all threads of the warp terminate. When all threads from the same warp execute the same instruction, they are executed concurrently - otherwise, some threads must wait. Therefore, the full efficiency of the computing platform may be obtained when all threads execute the same instruction in the same time, which means that conditional instructions significantly increase the computing time.

Experiments reported in this chapter were performed on a computing platform with the Intel Core i7 950 processor and two NVidia GeForce GTX 580 graphics cards, containing one many-core graphics processor with 16 multiprocessors each, connected by the Scalable Link Interface (SLI) bridge, but the approach should be compatible with other computing platforms based on the NVidia GeForce series supporting CUDA 3.2. Details of the hardware platform specification are presented in Table 5.2.

5.6.1 Parallel Architecture and Data Structures

Although evolutionary algorithms seem to be easy to run in parallel by splitting the entire reproduction process into a number of threads run at the same time on different processors, i.e. running the crossover and mutation operators in parallel, improvements in the computing time for such a parallel architecture are almost equal to costs of thread synchronization and data transmission.

Table 5.2 Hardware platform specification (NVidia GeForce GTX 580)

number of multithreaded streaming multiprocessors	16
number of logical cores per multiprocessor	32
total number of logical cores	512
number of registers per multiprocessor	16384
maximum number of threads per block	512
number of threads per warp	32
shared memory per multiprocessor	64 kB
constant memory	64 kB
local memory per thread	16 kB
maximum number of active blocks per multiprocessor	8
maximum number of active warps per multiprocessor	32
maximum number of active threads per multiprocessor	1024

However, in the case of rule selection, the most time-consuming part of the evolutionary algorithm is the financial simulation and the evaluation of the objective function, which may be easily run in parallel without any significant costs. Moreover, it is suitable for many-core graphics processors, because such parallel threads execute the same instructions in the same time by a majority of the computing time.

A number of technical issues had to be addressed, as a result of some limitations of the CUDA architecture. Due to memory constraints, financial time series of stock quotations were represented in short integer numbers (2 bytes) of euro cents. Simulations of trading expert performances were calculated in either integer numbers (4 bytes) or short integer numbers (2 bytes). Return rates and Sharpe ratios were calculated in float numbers (4 bytes, single precision arithmetic), but stored in integer numbers of 0.0001%, which in certain situations caused some numerical problems (insignificant in the final assessment). Permanent data structures, such as financial time series and trading rule signals, evaluated before the evolution had started, were stored in texture memory, as a result of the lack of a faster shared memory. Trading experts and objective values were stored in shared memory. Global memory was not used in computation, because of the low bandwidth.

5.6.2 Parallel Population Evaluation

Population evaluation was run in parallel – each trading expert from the population was processed in a separate thread. According to the CUDA architecture, threads was organized in blocks. Blocks were processed in parallel by multiprocessors, in such a way that a warp of 32 threads were processed at the same time, while the remaining warps of the same block were waiting active in the queue. The number of threads per block depended on the problem size, because only 64 kB of shared memory was accessible for the entire block (thus, the number of threads per block was approximately equal to 64 kB divided by the size of the trading expert and some temporary data structures necessary for the simulation). The stock price quotations and the precomputed signals of trading rules were stored in texture memory due to the limited size of the faster shared memory.

5.6.3 Parallel Local Search

Local search was also run in parallel: the Iterative Local Search operator optimized each trading expert in a separate thread, while the mainstream of the Iterative Local Search with Simulated Annealing operator was run on the sequential platform due to some inefficiency in parallel generation of random numbers. Parallelization of local search required a similar architecture to that of population evaluation, but a different number of threads was run.

Table 5.3 Comparison of the computing time necessary to construct a trading expert in the sequential and in the parallel approach (average times for 8 runs)

	parallel	sequential
1000 iterations, PBIL-RS without local search	50 s	2 min
5000 iterations, PBIL-RS without local search	240 s	9 min
1000 iterations, SGA-RS without local search	80 s	6 min
5000 iterations, SGA-RS without local search	390 s	29 min
1000 iterations, PBIL-RS with the ILS operator	110 s	7 min
5000 iterations, PBIL-RS with the ILS operator	540 s	34 min
1000 iterations, SGA-RS with the ILS operator	150 s	11 min
5000 iterations, SGA-RS with the ILS operator	740 s	53 min
1000 iterations, PBIL-RS with the ILS-SA operator	21 min	> 6 h
5000 iterations, PBIL-RS with the ILS-SA operator	1 h	> 6 h
1000 iterations, SGA-RS with the ILS-SA operator	24 min	> 6 h
5000 iterations, SGA-RS with the ILS-SA operator	1 h	> 6 h

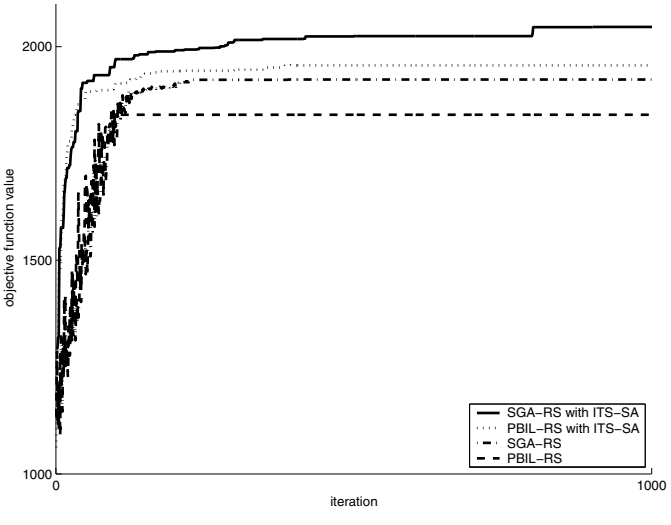


Fig. 5.1 The values of the objective function of the best trading expert discovered in the successive iterations of the evolutionary algorithm for the SGA-RS and the PBIL-RS algorithm and their modifications with the ILS-SA operator

5.7 Experiments

In order to validate the approach combining evolutionary algorithms with local search, a number of experiments were performed on a set of 500 trading rules based on technical analysis indicators [13] and 10 benchmark datasets. Each dataset consisted of financial time series of daily price quotations of one stock from the CAC IT 20 index of the Paris Stock Exchange over a training period from January, 2, 2009 to November, 30, 2009 (234 trading days) and a testing period from December, 1, 2009 to December, 31, 2009 (22 trading days). Performance of trading experts were evaluated in a financial simulation, with the simulation settings presented in Table 5.1, over the training period (in-sample) and the testing period (out-of-sample).

In the SGA-RS, the population size N was 3200, the offspring population size was $2M = 6400$, the cross-over probability p_c was 0.95, the mutation probability p_m was 0.05 and the evolution lasted $T = 1000$ or $T = 5000$ iterations. In the PBIL-RS, the population size N was 3200, the learning rate α was 0.15, the mutation probability β was 0.05, the mutation rate γ was 0.05 and the evolution lasted $T = 1000$ or $T = 5000$ iterations. In the ILS operator, neighbourhoods of size $\delta_1 = 4$ were considered. In the ILS-SA operator, mutations of size $\delta_2 = 6$ were used.

Each of the experiments described further was run twice: once on the sequential computing platform and once on the parallel computing platform. The sequential computing platform contained the Intel Core i7 950 processor. The parallel computing platform contained also two NVidia GeForce GTX 580 graphics cards connected by the Scalable Link Interface bridge with 1024 logical cores in total.

The first part of experiments aimed at comparing the computing time necessary to construct a trading expert in the sequential and in the parallel approach with many-core graphics

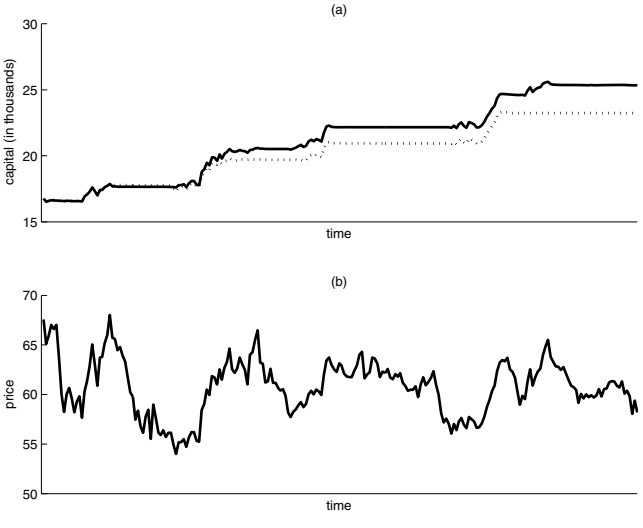


Fig. 5.2 The capital of the best trading expert discovered (a) and stock prices (b) in the successive days of the training period for an experiment with the dataset concerning Neopost

processors without focusing on the financial relevance of the investment strategies developed. Table 5.3 presents a summary of the comparison of the computing time. The first four rows correspond to the PBIL-RS and SGA-RS algorithms with local search turned off. The next four rows correspond to the algorithms with the Iterative Local Search operator. The last four rows correspond to the algorithms with the Iterative Local Search with Simulated Annealing (experiments on the sequential computing platform were stopped after 6 hours of computing). Not surprisingly, the parallel approach outperformed the sequential one in terms of the computing time and enabled to process the cases that were impractical for the sequential approach.

Figure 5.1 presents the values of the objective function of the best trading expert discovered in the successive iterations of the evolutionary algorithm for the SGA-RS and the PBIL-RS algorithm and their modifications with the ILS-SA operator. It is easy to see that local search significantly improves the standard evolutionary search.

The second part of experiments aimed at evaluating the financial relevance of the investment strategies discovered. It focused on the results of the parallel approach, because the financial relevance of both approaches is similar, the difference lies in the computing time, which sometimes makes the sequential approach impractical.

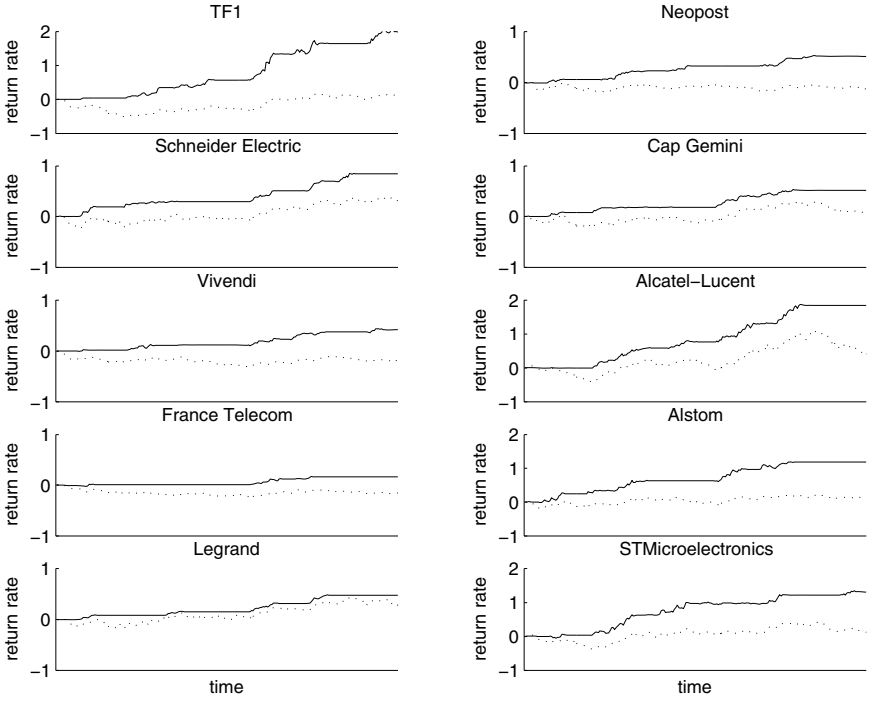


Fig. 5.3 A comparison of the capital of the best trading expert discovered (the solid line) with the capital of the simple Buy-and-Hold strategy (the dotted line) in the successive days of the training period for each dataset

Table 5.4 The financial relevance of the investment strategies discovered (average values for 8 runs)

Stock	ISIN	Training Time Time	Training Sharpe Ratio	Training Return Rate	Test Return Rate	Test B&H
Alcatel-Lucent	FR0000130007	24 min	28.45	289.13	6.46	5.84
Alstom	FR0010220475	19 min	21.13	93.84	5.69	4.92
Cap Gemini	FR0000125338	21 min	22.16	68.14	3.49	3.48
France Telecom	FR0000133308	14 min	12.03	20.24	0.35	0.24
Legrand	FR0010307819	18 min	24.17	74.36	1.83	6.09
Neopost	FR0000120560	22 min	20.39	50.84	0.65	-1.15
Schneider Electric	FR0000121972	23 min	26.14	92.47	17.37	11.91
STMicroelectronics	NL0000226223	34 min	21.94	83.96	5.99	19.73
TF1	FR0000054900	21 min	27.93	212.38	9.62	8.29
Vivendi	FR0000127771	26 min	17.46	73.14	8.87	8.03

Figure 5.2 presents the plot of the capital of the best trading expert discovered (the top subplot) and the plot of stock prices (the bottom subplot) over the training period for one chosen experiment with the dataset concerning Neopost (other experiments gave similar results and are summarized further). It is easy to see that the trading expert is more profitable than simple fluctuations in stock prices.

Figure 5.3 presents a comparison of the capital of the best trading expert discovered (the solid line) with the capital of the simple Buy-and-Hold strategy (the dotted line), which consists of investing the entire cash in stocks at the beginning and keeping it until the end of the considered period, over the training period for each dataset. It is easy to see that trading experts outperform the Buy-and-Hold strategy in all cases.

Finally, Table 5.4 presents the financial relevance of the investment strategies discovered. The first two columns define the dataset, the third column contains the computing time to find the best trading expert. The next two columns concern the training period and present the Sharpe ratio and the average return rate. The last two columns concern the testing period and present the return rate and the Buy-and-Hold benchmark. In all the experiments, the investment strategy discovered had a positive return rate. In most of cases, it also outperformed the Buy-and-Hold benchmark.

5.8 Conclusions

This chapter proposed hybrid evolutionary approaches, combining evolutionary algorithms with local search, for many-core graphics processors in order to improve the process of rule selection in stock market trading decision support systems. Parallelization of the sequential approach focused on population evaluation and local search. Despite some constraints of many-core graphics processors, mainly related to memory management, the parallel approach not surprisingly outperformed the sequential approach and enabled the processing of cases that were impractical for the sequential approach.

Although the main advantage of many-core graphics processors is the number of logical cores (in experiments, $2 \cdot 512 = 1024$ logical cores were used) and the number of threads run

in parallel, in order to achieve the high efficiency of computing, threads must execute similar instructions in the same time. Fortunately, in the decision support system discussed in this chapter, computation of the value of the objective function as well as the local search operators may be implemented in such a manner. Experiments carried out on real data from the Paris Stock Exchange confirmed that the approach proposed outperforms the classic approach also in terms of the financial relevance of the investment strategies discovered.

Further work on parallelization of more advanced evolutionary algorithms, such as the extended compact genetic algorithm (ECGA), BOA [17], or other Estimation of Distribution Algorithms [9], which may solve the optimization problem more efficiently, may make them competitive also in terms of the computing time.

References

1. Baluja, S.: Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning. Research Report CMU-CS-94-163 Carnegie Mellon University (1994)
2. Brabazon, A., O'Neill, M.: Biologically Inspired Algorithms for Financial Modelling. Springer, Heidelberg (2006)
3. Brandstetter, A., Artusi, A.: Radial Basis Function Networks GPU-Based Implementation. IEEE Transactions on Neural Networks 19, 2150–2161 (2008)
4. Chiosa, I., Kol, A.: GPU-Based Multilevel Clustering. IEEE Transactions on Visualization and Computer Graphics 17, 132–145 (2011)
5. Dang, J., Brabazon, A., O'Neill, M., Edelman, D.: Estimation of an EGARCH Volatility Option Pricing Model using a Bacteria Foraging Optimisation Algorithm. In: Natural Computing in Computational Finance, vol. 100, pp. 109–127. Springer, Heidelberg (2008)
6. Dempsey, I., O'Neill, M., Brabazon, A.: Adaptive Trading with Grammatical Evolution. In: Proceedings of the 2006 Congress on Evolutionary Computation (CEC 2006), pp. 2587–2592. IEEE Press, Los Alamitos (2006)
7. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley, Reading (1989)
8. Korczak, J., Lipinski, P.: Evolutionary Building of Stock Trading Experts in a Real-Time System. In: Proceedings of the 2004 Congress on Evolutionary Computation (CEC 2004), pp. 940–947. IEEE Press, Los Alamitos (2004)
9. Larranaga, P., Lozano, J.: Estimation of Distribution Algorithms, A New Tool for Evolutionary Computation. Kluwer Academic Publishers, Dordrecht (2002)
10. Lipinski, P.: Evolutionary Decision Support System for Stock Market Trading. In: Dochev, D., Pistore, M., Traverso, P. (eds.) AIMSA 2008. LNCS (LNAI), vol. 5253, pp. 405–409. Springer, Heidelberg (2008)
11. Lipinski, P.: Evolutionary Strategies for Building Risk-Optimal Portfolios. In: Natural Computing in Computational Finance. SCI, vol. 100, pp. 53–65. Springer, Heidelberg (2008)
12. Loraschi, A., Tettamanzi, A.: An Evolutionary Algorithm for Portfolio Selection within a Downside Risk Framework. In: Dunis, C.L. (ed.) Forecasting Financial Markets, pp. 275–286. Wiley, Chichester (1996)
13. Murphy, J.: Technical Analysis of the Financial Markets NUIF (1998)

14. Saks, P., Maringer, D.: Evolutionary Money Management. In: Giacobini, M., Brabazon, A., Cagnoni, S., Di Caro, G.A., Ekárt, A., Esparcia-Alcázar, A.I., Farooq, M., Fink, A., Machado, P. (eds.) EvoWorkshops 2009. LNCS, vol. 5484, pp. 162–171. Springer, Heidelberg (2009)
15. Sharpe, W.: Capital Asset Prices: A Theory of Market Equilibrium under Conditions of Risk. Journal of Finance 19, 425–442 (1964)
16. Tsang, E., Li, J., Markose, S., Er, H., Salhi, A., Iori, G.: EDDIE In Financial Decision Making. Journal of Management and Economics 4(4) (2000)
17. Wahib, M., Munawar, A., Munetomo, M., Akama, K.: A Bayesian Optimization Algorithm for De Novo ligand design based docking running over GPU. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2010), pp. 1–8. IEEE Press, Los Alamitos (2010)

6

Regime-Switching Recurrent Reinforcement Learning in Automated Trading

Dietmar Maringer and Tikesh Ramtohul

Universität Basel, CH-4002 Basel, Switzerland
{Dietmar.Maringer,Tikesh.Ramtohul}@unibas.ch

Summary. The regime-switching recurrent reinforcement learning (RSRRL) model was first presented in [19], in the form of a GARCH-based threshold version that extended the standard RRL algorithm developed by [22]. In this study, the main aim is to investigate the influence of different transition variables, in multiple RSRRL settings and for various datasets, and compare and contrast the performance levels of the RRL and RSRRL systems in algorithmic trading experiments. The transition variables considered are GARCH-based volatility, detrended volume, and the rate of information arrival, the latter being modelled on the Mixture Distribution Hypothesis (MDH). A frictionless setting was assumed for all the experiments. The results showed that the RSRRL models yield higher Sharpe ratios than the standard RRL in-sample, but struggle to reproduce the same performance levels out-of-sample. We argue that the lack of in- and out-of-sample correlation is due to a drastic change in market conditions, and find that the RSRRL can consistently outperform the RRL only when certain conditions are present. We also find that trading volume presents a lot of promise as an indicator, and could be the way forward for the design of more sophisticated RSRRL systems.

6.1 Introduction

The recurrent reinforcement learning (RRL), proposed by [22], is a direct reinforcement approach for investment decision making. It has an autoregressive outlook and can be likened to a recurrent neural network with a single layer. Previous work has already shown that the RRL offers good promise in finding profitable strategies in financial markets. Despite the reported findings, its simplistic nature casts some doubts about its ability to capture the non-linearities present in financial data. In [19], we proposed a new model, called threshold RRL, that augments the existing RRL with regime-switching properties using volatility as an indicator, to cater for these non-linearities. The main aim of this study is to investigate the more generic regime-switching RSRRL models (RSRRL) with different types of transition variables, and compare their performance with the basic RRL. We perform controlled experiments using artificial data to better understand the working principles of the algorithms, and then use real-world datasets to test the efficiency of the systems in a frictionless automated trading setting.

The outline of the paper is as follows: in Section 6.2, we present a review of previous work concerned with the application of RRL in financial trading. Section 6.3 is devoted to

A. Brabazon et al. (Eds.): Natural Computing in Comput. Finance, SCI 380, pp. 93–121.
springerlink.com

the RSRRL. It starts by briefly reviewing the RRL methodology and proceeds with a detailed description of the RSRRL model, with emphasis on the learning procedure. Section 6.4 then provides the motivation behind the choice of transition variables for this study, with the Mixture Distribution Hypothesis (MDH) taking centre stage. The ensuing section describes the experiments carried out to compare the two methodologies, presents the results, and provides an assessment of the main findings. Section 6.6 provides the concluding remarks and discusses possibilities for future work.

6.2 Literature Review

Early work by [22] and [23] aimed at demonstrating the efficiency of the RRL methodology for training trading systems and portfolios by optimising the differential Sharpe ratio (DSR). Their early studies emphasised on two main aspects. First, trading systems based on the reinforcement learning paradigm perform better than those based on supervised learning techniques. Second, mechanical traders trained to maximise a risk-adjusted performance criterion like the differential Sharpe ratio outperform trading systems trained on maximising profits alone, or on minimising some error criterion. Based on these results, [21] used real datasets to test the efficacy of the RRL-traders. They used the half-hourly U.S Dollar/British Pound FX rate from the first 8 months of quotes in 1996 to train a 3-position, i.e. $\{long, short, neutral\}$ trader. The differential downside deviation ratio [21] was used as the performance criterion. The RRL-traders led to profitable situations and positive Sharpe ratios, thereby indicating the ability of the RRL technique to discover structure in real-world financial data series. The authors also compared the performance of the RRL with a Q-learning approach, and found out that RRL-traders outperformed the Q-traders in all aspects, be it performance, interpretability or computational efficiency, which further enhanced the appeal for direct reinforcement learning approaches in the design of trading systems.

As a follow-up work on the single-layer RRL technique, [11] extended the model to a two-layer neural network and subsequently drew comparisons between the effectiveness of this variant with the original single-layer network. He used half-hourly quotes from 25 different FX markets for the entire year of 1996. The traders were of the $\{long, short\}$ type and the DSR was the objective function used for optimising the network weights. The author also performed some tuning to obtain good candidate values for some key model parameters like the learning rate and number of training epochs. His results showed that the RRL-traders were profitable in most markets, although for a select few, very low and even negative Sharpe ratios were reported. Despite the slightly mitigated performance, the general impression was that the RRL algorithm is able to capture certain patterns and come up with profitable situations. Moreover, the results also demonstrated that better performance is obtained with the one-layer network than with the two-layer version. The author attributed this to noisy financial data. He claimed that the more intricate version overfits the data and tentatively pointed out that trading in FX markets might not require models that are too complex.

A full-fledged automated trading system based on the RRL was put forward by [5]. They used a slightly modified version of the basic RRL as part of a trading system with a layered structure for trading in FX markets. The system consists of a machine learning layer, a risk management layer and a dynamic utility optimisation layer. The purpose of the risk management layer is to subject the output of the machine learning layer to certain risk constraints before the final trading decision is taken. The main role of the dynamic optimisation layer is to find optimal values for the model parameters in an adaptive fashion. They used one-minute data for the Euro-Dollar currency pair, spanning a period from Jan. 2000 up to Jan. 2002. The

results showed that the risk management layer and the dynamic utility optimisation layer give rise to better performance, hence implying that such a layered structure might be worth considering while designing fully automated trading systems. An important point reported by the authors concerns the use of inputs other than lagged returns to the RRL. They experimented with various popular technical indicators as input, but did not notice any added improvement in performance.

More recently, [2] used the RRL algorithm to develop a {*long*, *short*, *neutral*} trading system, and applied it to 9 of the major world financial market indices for the period between April 1992 and March 2007. The model is similar to the one proposed by [22] except that the authors used the reciprocal of the returns weighted direction symmetry index[1] as their maximisation criterion instead of the DSR. Daily closing prices were considered instead of high-frequency data. Moreover, a stop-loss criterion was included to prevent large drawdowns. Once more, results were very encouraging; the RRL-traders led to profitable situations in all but one case.

6.3 Model Description

6.3.1 Recurrent Reinforcement Learning

Reinforcement Learning (RL) is a type of machine learning technique which focuses on goal-directed learning from interaction [29]. It is a way of programming agents by reward and punishment without needing to specify how the task is to be achieved [13]; in other words, the learning process does not require target outputs, unlike supervised learning techniques. RL can be used to find approximate solutions to stochastic dynamic programming problems and it can do so in an online fashion [23]. In the last decade or so, it has attracted rapidly growing interest in the computational finance community, especially for the design of trading systems. The RRL, proposed by [22], is one such algorithm that uses the reinforcement paradigm to make investment decisions. It is an adaptive policy search algorithm which tries to maximise a certain performance criterion in order to learn profitable investment strategies. As its name suggests, the system is recurrent, meaning that the current investment decision has a say in shaping future decisions. In the presence of transaction costs, investment performance depends on sequences of interdependent decisions; the recurrent nature of the algorithm takes this path-dependency into account [23]. [21] describe the RRL as a computationally efficient algorithm that allows for simpler problem representation, avoids Bellman's curse of dimensionality, and circumvents problems that are generally associated with trading systems based on price forecasts.

The RRL model can be thought of as a gradient ascent algorithm which aims at optimising some desired criterion. The basic version was developed to trade fixed position sizes in a single security, but it can easily be extended to trade in varying quantities, or to manage multiple asset portfolios [23], or for asset allocation [23, 21]. A single-asset, two-position trader will be discussed in this paper. The trader can take only long or short positions of constant magnitude. Neutral positions are not allowed, so he is always in the market; this is also known as a reversal system [11]. The trading function is as follows:

[1] It corresponds to the ratio of the cumulative positive trading returns to the cumulative negative trading returns.

$$F_t = \tanh\left(\sum_{i=0}^{m-1} w_i r_{t-i} + w_m F_{t-1} + w_{m+1} v\right). \tag{6.1}$$

F_t is the output of the network at time t. A long position is adopted when $F_t > 0$; the trader buys an asset at time t and makes a profit if the price goes up in the next time step. If $F_t < 0$, the trader short sells an asset at time t and makes a profit if the price goes down at time $t+1$. The price return r_t corresponds to the difference in value of the asset between the previous period and the current period, i.e. $r_t = p_t - p_{t-1}$. The term v is the familiar bias present in neural network models, typically having a value of 1. The w_i's denote the system parameters or network weights that need to be optimised. Note that the time indexation of the weights has been dropped for clarity. The term F_{t-1}, i.e. the trade position at the previous time step, induces recurrence and hence some kind of internal memory. The RRL model is not restricted to taking only lagged price returns as input. It can easily accommodate technical indicators or other economic variables that might have an impact on the security.

6.3.2 Regime-Switching Recurrent Reinforcement Learning

Despite the relative success of the single-layer RRL model, it can be argued that its linear outlook makes it ill-suited to capture all the intricate aspects of financial data. An approach with a higher degree of non-linearity could very much aid in increasing its predictive capabilities. One straightforward way of accounting for the non-linearities is to incorporate hidden layers in the network. But, [11] noted a decline in performance when he introduced a hidden layer in the RRL topology. Indeed, multi-layer models are prone to overfitting, especially with noisy financial data, and are quite often unable to generalise properly. Moreover, such black-box approaches render inference about the input-output relationship difficult, if not impossible. A certain degree of transparency ensures that automated trading systems are more tractable, thereby allowing the human expert to adopt remedial measures or perform fine-tuning more efficiently whenever performance starts to degenerate. There is a need for non-linear models that can perform well out-of-sample and that can shed some light on how economic variables affect financial markets. Regime-switching models provide an elegant solution to this kind of problem. These models define different states of the world (regimes), and assume that the dynamic behaviour of economic variables depends on the regime that occurs at any given point in time. This implies that certain properties of the time series, such as its mean, variance, autocorrelation, etc., are different in different regimes. Such models offer a great deal of transparency and the concept of regimes helps to capture non-linearities.

There exists some well-established regime-switching methods that have gained prominence in econometrics. These include the threshold model, initially proposed by [32], the Markov-Switching model of [12], the artificial neural network model of [34], and the smooth transition model [31], the latter being a more general version of the threshold model. Suppose that we have a 2-regime situation for some dependent variable x_t, and each regime is characterised by an $AR(1)$ process; then the regime-switching model can be expressed as

$$\begin{aligned} x_t = &\left(\phi_{0,1} + \phi_{1,1} x_{t-1}\right)\left(1 - G(q_t;\gamma,c)\right) \\ &+ \left(\phi_{0,2} + \phi_{1,2} x_{t-1}\right) G(q_t;\gamma,c) + z_t, \end{aligned} \tag{6.2}$$

where z_t denotes an i.i.d white noise process. $G(q_t;\gamma,c)$ is the transition function, for which the logistic function is a popular choice and takes the form

$$G(q_t;\gamma,c) = \frac{1}{1+\exp\left(-\gamma[q_t - c]\right)}, \tag{6.3}$$

where q_t is the transition variable, c is the threshold or location parameter, and γ dictates the smoothness of the transition. $G(q_t;\gamma,c)$ can take any value in the range $[0,1]$. As γ tends to infinity, the logistic function approaches the indicator (step) function. The interested reader is referred to [10] for more details about these models.

The regime-switching version of the recurrent reinforcement algorithm can be formulated by considering (6.1) and (6.2). To simplify the discussion, the focus will be on models that involve only two regimes. It is however trivial to extend the model to account for multiple regimes and/or multiple indicator variables. A two-regime system ($j = \{1,2\}$) can be described as

$$F_t = y_{t,1} G(q_t;\gamma,c) + y_{t,2}\left(1 - G(q_t;\gamma,c)\right), \tag{6.4}$$
$$y_{t,j} = \tanh\left(\sum_{i=0}^{m-1} w_{i,j} r_{t-i} + w_{m,j} F_{t-1} + w_{m+1,j} v\right).$$

These systems can be thought of having two RRL networks (see Figure 6.1), each one corresponding to a particular regime and having a distinct set of weights. The overall output F_t of the system is the weighted sum of the the outputs $y_{1,t}$ and $y_{2,t}$ of the individual networks. The weighting factor is actually the value of the transition variable. Initially, both networks have the same set of weights. During training, the model promotes selective learning and this leads to each network developing a unique set of weights. If the system is in a particular regime, the network associated with that regime is exposed to higher weight updates than the other. The extent of the overlapping between the two regimes is regulated by the term γ. For the threshold version ($\gamma = \infty$), each network learns a distinct mapping that corresponds to a specific region in the space spanned by the indicator variable. The latter effectively acts as a switch or gating device that selects the appropriate network at each time step.

6.3.3 Differential Sharpe Ratio for Online Learning

The learning process of the RSRRL is in essence similar to that of the RRL described in [22]. It involves maximising a certain performance criterion to obtain a set of network weights that can lead to profitable strategies. [23] showed that RRL systems trained by maximising risk-adjusted performance criteria perform better than those trained by minimizing error functions. They used stochastic gradient ascent to maximise the differential Sharpe ratio (DSR), a variant of the well-known Sharpe ratio introduced by [27]. The DSR is derived by making use of exponential moving average estimates of the first and second moments of the trading returns distribution. The same approach has been adopted in this paper. The trading return R_t, as defined by [23], is expressed as

$$R_t = r_t^f + \operatorname{sign}(F_{t-1})\left(r_t - r_t^f\right) - \delta \left|\operatorname{sign}(F_t) - \operatorname{sign}(F_{t-1})\right|, \tag{6.5}$$

where r_t^f is the risk-free rate of interest and δ is the transaction cost rate per share traded . Note that both of these terms have been assumed to be zero in this study. The exponential moving average Sharpe ratio can be expressed in terms of the trading return R_t. It is given by

$$S_t = \frac{A_t}{\sqrt{B_t - A_t^2}}, \tag{6.6}$$

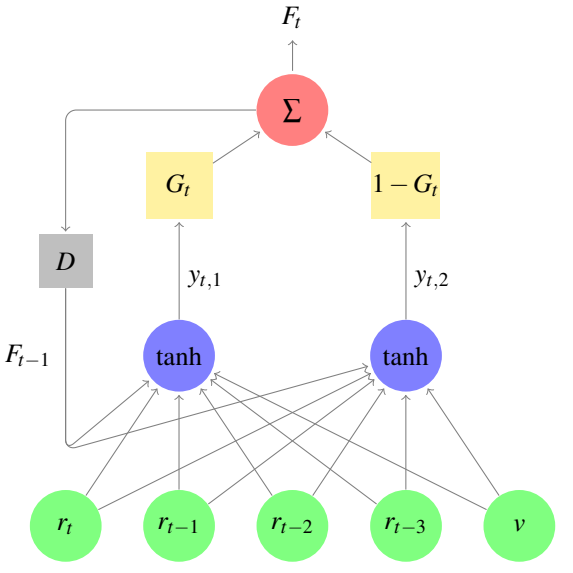


Fig. 6.1 RSRRL network structure where D is the delay operator and $m = 4$

where

$$A_t = A_{t-1} + \eta\,(R_t - A_{t-1}) = A_{t-1} + \eta \Delta A,$$
$$B_t = B_{t-1} + \eta\left(R_t^2 - B_{t-1}\right) = B_{t-1} + \eta \Delta B.$$

The DSR is obtained by expanding the exponential moving average version to first order in the adaptation rate η [23]. It is given by

$$D_t = \frac{B_{t-1}\Delta A - \frac{1}{2}A_{t-1}\Delta B}{\left(B_{t-1} - A_{t-1}^2\right)^{\frac{3}{2}}}. \tag{6.7}$$

It can be optimised incrementally using gradient ascent. If ρ corresponds to the learning rate, the weight update equation is given by

$$w_{t,j} = w_{t-1,j} + \rho \Delta w_{t,j} \qquad \text{for } j = \{1,2\}, \tag{6.8}$$

where

$$\Delta w_{t,j} = \frac{dD_t}{dR_t}\left(\frac{dR_t}{dF_t}\frac{dF_t}{dw_{t,j}} + \frac{dR_t}{dF_{t-1}}\frac{dF_{t-1}}{dw_{t-1,j}}\right).$$

The derivative $\frac{dF_t}{dw_t}$ for online training can be computed using an approach similar to back-propagation through time (BPTT) introduced by [33] and discussed in [23],

$$\frac{dF_t}{dw_{t,j}} \approx \frac{\partial F_t}{\partial w_{t,j}} + \frac{\partial F_t}{\partial F_{t-1}}\frac{dF_{t-1}}{dw_{t-1,j}} \qquad \text{for } j = \{1,2\} \tag{6.9}$$

where

$$\frac{\partial F_t}{\partial w_{t,j}} = \frac{\partial F_t}{\partial y_{t,j}} \times \frac{\partial y_{t,j}}{\partial w_{t,j}},$$
$$\frac{\partial F_t}{\partial F_{t-1}} = \sum_{j=1}^{2} \left(\frac{\partial F_t}{\partial y_{t,j}} \times \frac{\partial y_{t,j}}{\partial F_{t-1}} \right).$$

All the required derivatives can be computed using basic differentiation rules, and thus the weight update process turns out to be rather straightforward and relatively fast.

6.4 Transition Variables

6.4.1 General Considerations

There exists a host of indicators that are typically used in technical analysis for predicting price movements, but only a few can actually be regarded as potential candidates for switching between regimes in the RSRRL model. The very nature of the RSRRL calls for a transition variable that has certain desirable characteristics. First and foremost, it must have an impact on the serial correlation of the price returns process. This is a rather obvious requirement since the model takes lagged returns as input and is thus sensitive to the autocorrelations present in the data. Absence of any relationship between the indicator variable and serial correlation in the returns will most certainly lead to spurious learning. Another important feature of the transition variable is that it should be computable or observed at a frequency that is at least equal to the desired frequency of trading. For instance, if data about a certain economic factor is reported only once per month, it cannot be used in a system designed for daily trading. A third feature is concerned with the switching behaviour of the variable. Excessive switching during the training phase will prevent the networks from uncovering structure in the data since proper learning requires sustained weight updates in any of the regimes. On the other hand, little or zero switching behaviour will cause learning to occur in only one of the networks. It is therefore imperative to have a variable that spans a regime over multiple time steps. On top of that, during the training phase, it is also important that the system goes through enough instances of each regime so that learning is not biased towards one network. Finally, it should preferably be an observable variable or a function of an observable variable that can be readily computed. Latent variables can also be good candidates as long as the estimation process is fast and reliable. Otherwise, the model runs the risk of having estimation errors propagating during the learning phase that can undermine its generalisation capabilities.

The transition variables considered in this study are volatility, trading volume and the daily rate of information arrival. They have most of the aforementioned characteristics and can therefore be considered as good candidates for the RSRRL. The ensuing subsections review some empirical studies regarding these variables, and presents the motivation behind their use in the regime-switching framework.

6.4.2 Volatility

Empirical studies have shown that volatility and serial correlation in returns are related. [17] used a GARCH model with an exponential time varying first order autocorrelation to describe the short run dynamics of several US stock index returns as well as individual stock returns.

His results pointed at larger first-order autocorrelations during periods of lower volatility than during periods of higher volatility. These patterns were observed in both daily and weekly returns. [26], in turn, found out that during low volatility periods, daily US index returns are positively autocorrelated but during high volatility periods, they tend to be negatively autocorrelated. The authors attribute this finding to the use of feedback strategies during trading, suggesting that the observed level of autocorrelation at any given point in time is a function of the strength of positive feedback trading relative to negative feedback trading. According to them, the sign reversal in stock return autocorrelation arises because positive feedback traders exert a greater influence on price movements and the degree of autocorrelation during volatile periods. [15] investigated this angle further by considering national stock markets other than the US. He also found evidence of positive feedback trading causing stock return autocorrelation to become negative during high volatility periods. He also noted that positive feedback trading is more pronounced during market declines than it is during market advances. And finally, [20] investigated this relationship on US stock returns using a bivariate GARCH model with time-dependent variance and autocovariance. They found that either positive or negative autocorrelation exists at low volatility levels, but an increase in volatility increases the likelihood of negative autocorrelation. These studies do not provide a unified view about the relationship between volatility and serial correlation, but they nevertheless provide enough justification for the use of volatility as a transition variable.

6.4.3 Volume

Over the last few decades, the research community has spent a lot of time and effort in trying to understand the nature of connections between volume and prices of securities. According to [14], good understanding of these relationships can provide valuable insight to the structure of financial markets by providing information regarding rate of information flow in the marketplace and the extent that prices reflect public information. In particular, the relationships between volume and volatility, and between volume and price returns, have been scrutinised. [14] conducted a comprehensive survey on early empirical work in this area, and formulated a couple of 'stylized facts', namely

- The correlation between volume and the absolute value of the price change (a kind of volatility measure) is positive.
- The correlation between volume and the price change *per se* is positive.

The first proposal has almost unanimously been accepted (see [14] for a summary of the main empirical studies in this area), with the main debate now centering on whether the volume-volatility relationship is causal or contemporaneous. The second proposal, however, is still subject to much debate. It literally implies that volume is relatively heavy in bull markets but light in bear markets [14]. [36] was among the first to postulate such a relationship, based on a series of statistical tests conducted on a six-year daily series of price and volume. He reported a set of empirical findings, of which the following two statements are of pertinent interest to this study

- A small volume is usually accompanied by a fall in price.
- A large volume is usually accompanied by a rise in price.

Subsequent work by [7] and [8] supported these findings: they found that the ratio of volume to absolute price change was larger for transactions on upticks than on downticks, both in the stock and bond markets. However, conflicting evidence was found by [35] who found the ratio to be higher for downticks. There are other empirical studies, reported by [14] in his seminal

paper, that tend to either agree or disagree with the findings of [36]. This has led to a divided opinion among researchers today about the true nature of this relationship. Nevertheless, it remains an interesting candidate for use as indicator variable, especially since volume data is readily available and can easily be processed.

6.4.4 Rate of Information Arrival

Among the many factors that affect the amount of daily volume on a security, the arrival of new information arguably has the greatest impact of them all. New information, or news, can be a press release or a regular earnings announcement provided by the company, or it can be a third party communication, such as a court ruling or a release by a regulatory agency pertaining to the company [28]. It is quite natural then that most of the theoretical models that have been put forward to help explain the price-volume relationships are based on the arrival of new information. One such model is the Mixture Distribution Hypothesis (MDH). [30] proposed a market microstruture model that led to the formulation of the MDH. In their model, the market consists of J active traders. Within a day, the market for a single security passes through a sequence of distinct equilibria. The movement from the $(i-1)^{th}$ to the i^{th} within-day equilibrium is initiated by the arrival of new information to the market. The number of traders J is non-random and fixed for each day, while the number of daily equilibria, I, is random. At the i^{th} equilibrium the desired position of the j^{th} trader is given by $q_{ij} = \alpha(p^*_{ij} - p_i)$, where p^*_{ij} is his reservation price, p_i is the current market price and α is a positive constant. Under the equilibrium condition that the market clears, the market price is determined by the average of the reservation prices of all J traders.

$$\sum_{j=1}^{J} q_{ij} = 0 \qquad \Rightarrow \qquad p_i = \frac{1}{J}\sum_{j=1}^{J} p^*_{ij}. \tag{6.10}$$

The influx of new information causes the traders to adjust their reservation prices, which consequently causes a change in the market price. The associated volume of trading v_i is by definition one-half the sum of the absolute of the changes in the traders' positions. Hence,

$$\delta p_i = p_i - p_{i-1} = \frac{1}{J}\sum_{j=1}^{J} \delta p^*_{ij} \tag{6.11}$$

$$v_i = \frac{\rho}{2}\sum_{j=1}^{J} |\delta p^*_{ij} - \delta p_i| \tag{6.12}$$

The authors used the following variance-components model to express the change in the reservation prices of the traders,

$$\delta p^*_{ij} = \phi_i + \psi_{ij} \qquad \text{with} \qquad \phi_i \sim \text{i.i.d.} N(0, \sigma^2_\phi), \qquad \psi_{ij} \sim \text{i.i.d.} N(0, \sigma^2_\psi). \tag{6.13}$$

where ϕ_i represents the component common to all traders and ψ_{ij} represents the idiosyncratic component specific to the j^{th} trader. The parameters σ^2_ϕ and σ^2_ψ measure the sensitivity of the traders' reservation prices w.r.t. new information. A large realization of ϕ_i relative to ψ_{ij} depicts the case where traders are unanimous about the new information. The alternate scenario means that traders react diffusely to the news. In this model, the variances of ϕ_i and ψ_{ij} are time independent. The i^{th} market price change and trading volume, based on the variance-component model is given by:

$$\delta p_i = \phi_i + \overline{\psi_i}, \quad \text{with} \quad \overline{\psi_i} \equiv \frac{1}{J}\sum_{j=1}^{J} \psi_{ij} \tag{6.14}$$

$$v_i = \frac{\alpha}{2}\sum_{i=1}^{I} |\psi_{ij} - \overline{\psi_i}| \tag{6.15}$$

Thus, for a given day t, the daily price change, δp_t or r_t, and the daily trading volume, v_t, can be obtained by summing up the within-day price changes, δp_i, and intra-day volume v_i. [30] derived the first and second moments for r_t and v_t, and proposed that the joint distribution of daily returns and trading volume is a bivariate normal conditional on the daily number of information arrivals I_t if it is assumed that the number of traders J, as well as the variances σ_ϕ^2 and σ_ψ^2 (see (6.13)), are constant over time. Their model boils down to the following equations:

$$\begin{aligned} r_t | I_t &\sim N(0, \sigma_r^2 I_t), \\ v_t | I_t &\sim N(\mu_v k_t, \sigma_v^2 I_t). \end{aligned} \tag{6.16}$$

The return variance and trading volume are simultaneously directed by the rate of information flow to the market, thereby implying a positive contemporaneous relationship between volatility and volume. The dynamics of the volatility process is dependent on the time series behaviour of I_t, which also affects the trading volume. The authors assumed a log-normal distribution for I_t,

$$\lambda_t \sim N(\mu_\lambda, v_\lambda^2) \quad \text{where} \quad \lambda_t = \log(I_t). \tag{6.17}$$

Since the behaviour of the latent process directly influences volatility dynamics, many researchers focused their attention on the MDH in a bid to explain ARCH effects in financial time series. [16] assumed the information arrival rate to be serially correlated and used a Gaussian $AR(1)$ process to model this. [1] then extended the model by including an additional component in the volume equation, based on a microstructure framework he proposed. The component in question is unrelated to information flow, but rather reflects noise or liquidity trading. His model can be represented by the following set of equations

$$\begin{aligned} r_t | \lambda_t &\sim N(0, \sigma_r^2 e^{\lambda_t}), \\ v_t | \lambda_t &\sim N(\mu_0 + \mu_w e^{\lambda_t}, \sigma_v^2 e^{\lambda_t}), \\ \lambda_t | \lambda_{t-1} &\sim N(\delta_\lambda \lambda_{t-1}, v_\lambda^2). \end{aligned} \tag{6.18}$$

Other notable contributors in this area are [18] and [9] who extended the model of [1] in an attempt to adequately reflect the joint behaviour of volatility and volume, but none has been able to fully account for persistence in stock price volatility. The MDH might have certain shortcomings, but it is in agreement with many of the empirical findings discussed earlier. The latent directing variable, i.e. the rate of information arrival, makes a very strong case for its use as indicator variable in the RSRRL.

6.5 Experiments

6.5.1 Setup

This section describes the experiments carried out to gauge the efficiency of the RSRRL model. The simulation results are presented and an assessment of the main findings is given.

The first experiments dealt with artificial data series and aimed at illustrating the capabilities and potential downsides of the RSRRL. The second string of experiments aimed at comparing the abilities of the RSRRL models to discover structure in real financial price series, using different transition variables and different slope parameters.

The traders studied were of the $\{long, short\}$ type who could buy/sell only 1 share at a time. If a trader is already in a certain position, he holds this position until the reverse trade signal is output by the system. Transaction costs and market impact were assumed to be zero. The training phase consisted of subjecting the traders to training data of length L_{tr} for a number of epochs n_e. The trades made during the training period allow the systems to update their weights in a bid to generalise properly when faced with novel data. The trained networks were then subjected to an ensuing out-of-sample period L_{te}.The traditional Sharpe ratio was used as performance measure. It is to be noted that the weight update process continues throughout the test period.

The model parameters include the learning rate ρ, the adaptation rate η, the number of price return inputs m, the size of the training window L_{tr},the number of training epochs n_e, and the size of the test window L_{te}. The values used were $L_{tr} = 2000$, $L_{te} = 375$, $m = 5$, $n_e = 5$, $\rho = 0.01$, and $\eta = 0.01$, inspired from previous work by [21] and [11]. The initial weights were sampled from a Gaussian distribution with a mean of 0 and a standard deviation of 0.1.

Because of the non-stationarity of the objective function, the optimisation process is not very stable. The models are very sensitive to the initial weights. Preliminary experiments need to be carried out to obtain good values for the initial weights. Alternatively, the best performing traders during the in-sample period could be picked to be candidates for the out-of-sample period. It is reasonable to assume that these traders will perform better during the test period than the worst in-sample performers. This approach has been adopted for this paper. A bunch of traders were trained and the Sharpe ratio achieved by each trader at the end of training was recorded. Only the top $x\%$ performers (referred to as 'elitists' hereafter) were selected for the testing phase. The in-sample performance was taken to be the mean of these elitists. The final out-of-sample trades were generated by pooling the individual signals of these elitists (using a majority rule), in an attempt to minimise overfitting problems that might have crept in during training. Such a procedure, both for the training and testing phases, ensures that results are reproducible, meaning that the use of different RNG seeds for the weights will not lead to distinctly different performance levels.

6.5.2 Artificial Data

A set of controlled experiments was conducted using artificial return series to compare and contrast the behaviour of the RSRRL traders with respect to the RRL traders. Five scenarios were considered, in an attempt to have a good understanding of the working principle of the regime-switching model. The generic form of the data-generating mechanism was as follows

$$r_t = \begin{cases} \phi_{1,1} r_{t-1} + \phi_{2,1} r_{t-2} + z_t & \text{if } D_t = 1 \\ \phi_{1,2} r_{t-1} + \phi_{2,2} r_{t-2} + z_t & \text{if } D_t = 0, \end{cases} \tag{6.19}$$

where D_t is a an artificially generated binary variable and z_t is $i.i.d$ white noise. For the first scenario, both regimes were generated by AR processes with identical coefficients, effectively corresponding to a single-regime situation. The coefficients used were ($\phi_{1,1} = \phi_{1,2} = -0.2$, $\phi_{2,1} = \phi_{2,2} = -0.1$). For the second scenario, the data series were constructed from the concatenation of two independent AR processes exhibiting autocorrelation of same magnitude

but of different sign. In other words, the data from scenario 2 is made up of portions having either negative autocorrelation or positive autocorrelation. The coefficient set for this scenario was ($\phi_{1,1} = 0.2$, $\phi_{2,1} = 0.1$; $\phi_{1,2} = -0.2$, $\phi_{2,2} = -0.1$). The regimes are determined by the indicator values which are generated randomly but with a restrictive switching frequency. In the next one, the dataset from the second scenario was used; however, another indicator function was used for partitioning the data during learning, meaning that spurious regime information was fed to the RSRRL system. In the fourth scenario, AR processes with the same sign, but having different magnitude were considered ($\phi_{1,1} = -0.4$, $\phi_{2,1} = -0.2$; $\phi_{1,2} = -0.2$, $\phi_{2,2} = -0.1$). The final case looked at return series that consisted of distinctly different training and test periods, in the sense that the AR coefficients for the out-of-sample phase did not match those used for the in-sample period. The coefficient set for the test period was ($\phi_{1,1} = 0.1$, $\phi_{2,1} = -0.2$; $\phi_{1,2} = 0.2$, $\phi_{2,2} = -0.1$), while the set from the second scenario was used for the training part. For each scenario, 500 different realisations of the respective stochastic process were generated, and for each realisation, 10000 different sets of initial network weights were used for training. For each individual case, the top 100 in-sample performers (99^{th} quantile) were selected for generating the pooled out-of-sample trades.

Table 6.1 Artificial Data Results

Scenario	Sample	$R_1 > 0$	$R_2 > 0$	$R_2 > R_1$	$E(R_2 - R_1)$
First	In	100.0 %	100.0 %	66.8 %	0.0054*
	Out	100.0 %	100.0 %	45.0 %	−0.0043
Second	In	100.0 %	100.0 %	100.0 %	0.0901*
	Out	63.0 %	100.0 %	97.4 %	0.1278*
Third	In	100.0 %	100.0 %	83.8 %	0.0138*
	Out	63.0 %	58.8 %	47.8 %	−0.0047
Fourth	In	100.0 %	100.0 %	57.0 %	0.0014*
	Out	100.0 %	100.0 %	49.8 %	−0.0020
Fifth	In	100.0 %	100.0 %	100.0 %	0.0901*
	Out	47.4 %	27.8 %	36.0 %	−0.0270

R_1 corresponds to the standard RRL, R_2 to the RSRRL. Scenario 1 is for a single-regime situation, scenario 2 for regimes with different sign but same magnitude, scenario 3 is similar to the second scenario but deals with incorrect regime identification, scenario 4 is for regimes with same sign but different magnitude, and finally scenario 5 depicts different regime configurations in the training and test period.

The results for this set of experiments are summarised in Table 6.1. R_1 corresponds to the standard RRL, R_2 to the RSRRL. The absolute and relative performances of each type of trader are reported, as well as the t-test results with alternative hypothesis that $R_2 > R_1$ (significant results at the 5% level are marked by the * symbol). In the first scenario, both types of systems yield positive Sharpe ratios for all cases studied, in the in-sample as well as the out-of-sample period. The RSRRL perform better than the RRL during the training phase; it achieves higher Sharpe ratios in 334 out of 500 cases (66.8%). The average difference in

Sharpe ratios is however very small, as indicated in the last column of Table 6.1. The situation is reversed in the test phase during which the RRL on average fares marginally better. It seems that both networks of the RSRRL system are able to uncover the true data generating mechanism. The corresponding weights in each branch typically have the same sign, although they might differ in magnitude. Thus, both branches tend to produce very similar trade signals after training, and the overall output of the RSRRL becomes almost regime-independent.

The results for the second scenario indicate that the RSRRL completely outperform the RRL, both in-sample and out-of-sample. Both systems achieve positive Sharpe ratios during training for all the cases studied, with the RSRRL performing better in each case. During the test phase, the RSRRL traders remain profitable for all the different configurations, while a good proportion of the RRL-traders (37%) struggle and end up with negative Sharpe ratios. The RSRRL achieves higher Sharpe ratios than the RRL in 97.4% of the cases studied during the out-of-sample period. These results suggest that the standard RRL cannot properly deal with datasets in which the serial correlation changes sign from one portion to another. It seems that such datasets have a nullifying effect on the learning process. A network with a single set of weights cannot perform well in those two distinctly different regimes. Whenever there is a regime shift, the RRL takes time to adjust to its new environment and is unable to come up with profitable strategies. The RSRRL model, on the other hand, is able to avoid this pitfall, since it develops a specific set of weights for each regime.

In the third scenario, it can be seen that the RSRRL systems are not able to outperform the RRL systems like they did in the second case. Interestingly though, despite the incorrect regime information given to the regime-switching systems, they are able to achieve positive Sharpe ratios in more than half of the simulations during the out-of-sample period. There is an amount of overlap that exists between both sets of indicators used in this scenario; for certain periods in time, the regimes would be correctly identified, and this aids the RSRRL in uncovering some structure in the data.

For the fourth case, the results are quite similar to those obtained for the first scenario. The figures indicate that it is very difficult, *a priori*, to pick up a winner between the two systems in single-regime situations, or with regimes that are closely related to each other, e.g., when the regime-dependent return series have serial correlation of the same sign.

In the final scenario, both systems struggle in the out-of-sample periods and end up making losses in the majority of cases. The RRL systems however perform better than their RSRRL counterparts in general. There are a number of conclusions that can be drawn from the results obtained with the artificial datasets. In single regime situations, or with regimes that are closely related to each other, the RSRRL model will on average match the performance of the RRL. However, the RSRRL system appears to be more prone to overfitting, as suggested by the results in the first scenario. This is probably due to the selective weight updating in the sub-networks of the threshold model, which implies that the sub-networks are typically exposed to a smaller region of the input space, compared to the standard RRL network. Thus, each subnetwork has a smaller data pool to learn from, and is therefore more likely to be influenced by outliers. In the presence of regimes having distinctly different correlation levels, the RSRRL is more profitable than the RRL as long as the regimes are correctly identified. In case of the advent of completely new market conditions, the performance of the RSRRL is more severely affected than that of the RRL, which suggests that the latter is more robust than the regime-switching approach. The RSRRL, and more specifically the TRRL, invariably suffers if the test period is completely different from the training period, since it is exposed to regimes that have never been encountered before. Moreover, since the sub-networks might already be having some overfitting issues, the RSRRL networks take longer to adapt to their new surroundings and are unable to adjust their weights properly to cope with the altogether

new regimes. Of course, in certain situations, the RSRRL systems can be more adapted to drastic changes in market conditions than the RRL, e.g., if a certain regime that was predominantly present in the first half of the training period comes back in force during the test period. Overall, it seems that the RSRRL does not guarantee superior performance over the standard RRL, and is heavily dependent on the data structure, more so than the standard RRL.

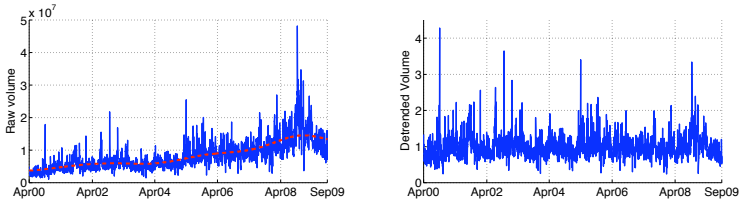


Fig. 6.2 Raw and Detrended volume data for CVX. The stochastic trend component is represented by the dashed line.

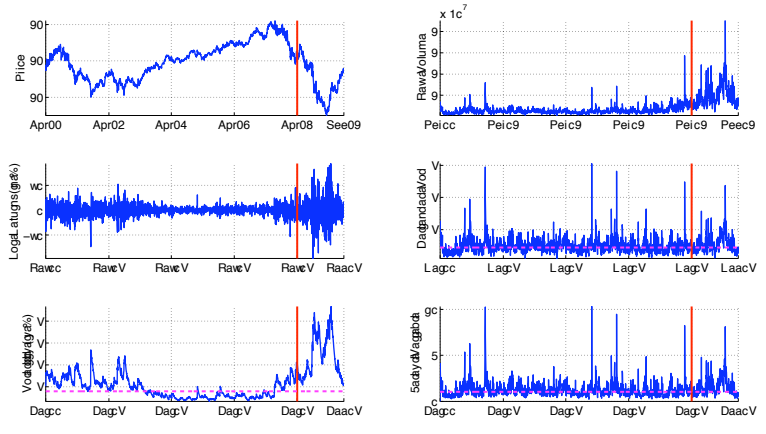


Fig. 6.3 The time series associated with the AXP data. The dashed horizontal lines associated with the graph of the transition variables correspond to the threshold values.

6.5.3 Real Financial Datasets

Fifteen components of the Dow Jones Industrial Average (DJIA) index were considered for the the automated trading experiments. These are American Express (AXP), Boeing (BA), Bank of America (BAC), Chevron Corporation (CVX), Dupont (DD), General Electric (GE), International Business Machines (IBM), JPMorgan Chase (JPM), 3M (MMM), Procter & Gamble (PG), AT&T (T), United Technologies Corporation (UTX), Verizon Communications (VZ), Wal-Mart (WMT), ExxonMobil (XOM). A nine-year period from April 2000 upto September 2009 was considered. It was divided into a training portion consisting of 2000 datapoints (roughly 8 years of data), and a test portion of 375 datapoints corresponding to the period between April 2008 and September 2009. From an economic viewpoint, the

datasets reflect chronologically the end of the dot.com bubble, followed by the start of the US housing bubble and its subsequent deflation that culminated into the financial crisis of 2008. The daily log return series (in %) of these stocks constituted the dataset for the RRL models. Regarding the transition variables, a GARCH(1,1) model was used for extracting the volatility estimates, the MDH model by [1] was chosen for generating the estimates of the rate of information arrival, and a detrending technique based on a stochastic trend component was applied to the raw volume data to produce a stationary volume series. Note that for all the transition variables considered, the threshold c in the transition function was set equal to the median of the distribution of the respective variable over the training period. Figure 6.2 illustrates this transformation for the CVX dataset. Figure 6.3 shows all the time series of interest for the AXP data. The dashed horizontal lines correspond to the thresholds, while the vertical line demarcates the test period from the training period. A more detailed description about the transition variables is presented in the appendix.

As a pre-analysis tool, the nature of the serial correlation (if any) in the log return series associated with the regime-switching systems was investigated. The following threshold model was fitted to the datasets for both the training and test portions,

$$r_{t+1} = (\boldsymbol{\Phi}_2 + (\boldsymbol{\Phi}_1 - \boldsymbol{\Phi}_2) I(q_t > c)) \mathbf{R_t} \tag{6.20}$$

where

$$\begin{aligned} \boldsymbol{\Phi}_1 &= \left[\phi_{0,1}, \phi_{1,1}, \ldots, \phi_{5,1}\right], \\ \boldsymbol{\Phi}_2 &= \left[\phi_{0,2}, \phi_{1,2}, \ldots, \phi_{5,2}\right], \\ \mathbf{R_t} &= \left[1, r_t, r_{t-1}, \ldots, r_{t-4}\right]^T. \end{aligned}$$

The coefficient sets $\boldsymbol{\Phi}_2$ and $\boldsymbol{\Phi}_1 - \boldsymbol{\Phi}_2$ were estimated using OLS regression. For the in-sample period, the volume-based and latent variable-based datasets yielded significant results, while the coefficients for the GARCH-based datasets were mostly insignificant. The volume-based coefficients for the training period are given in Table 6.2. The coefficients for the latent variable-based series have not been reported for brevity purposes, but they are by and large quite similar to the volume-based coefficients. The GARCH-based coefficients have also been omitted because of the lack of statistical significance. The volume-based results reveal that, for all the datasets, the first-order coefficients are significant, and in quite a few cases, hint towards the presence of regime-dependent autocorrelation levels that differ in both sign and magnitude.

For the out-of-sample period, the results are quite different. The vast majority of the coefficients were found to be statistically insignificant, even for the volume- and latent-based systems. For comparative purposes, the volume-based coefficients for the out-of-sample period are reported in Table 6.3. The estimated coefficients for both the training and test periods can help provide some insight into the performance of the RSRRL systems, by virtue of their autoregressive construction.

The experimental setup for the real-world datasets was as follows. There were 6 types of traders considered for the in-sample period, namely the plain RRL traders (RRL), the GARCH-based RSRRL traders (GRRL), the volume based traders (VRRL), the latent variable based traders (LRRL), the GARCH/volume based traders (GVRRL), and random traders (RND). As their name suggests, the latter trade randomly, generating buy or sell signals with equal probability. They have been included for benchmarking purposes. The GVRRL consisted of two RSRRL systems, one having GARCH estimates as transition variable and the other having the detrended volume values as transition variable. They were allowed to learn

Table 6.2 Volume-dependent AR coefficients for in-sample period

	AXP	BA	BAC	CVX	DD	GE	IBM	JPM	MMM	PG	T	UTX	VZ	WMT	XOM
Φ_2	0.0065	0.0680	0.0460	0.1815*	−0.0108	−0.0451	−0.0367	0.0011	0.0504	0.0404	0.0038	0.1570*	−0.0240	−0.0395	0.1238*
	−0.1755*	−0.2181*	−0.1515*	−0.1743*	−0.1738*	−0.1779*	−0.1772*	−0.1352*	−0.1462*	−0.1467*	−0.1419*	−0.2772*	−0.1735*	−0.1587*	−0.1885*
	−0.1114*	−0.0593	−0.0143	−0.1006*	−0.0464	−0.1156*	−0.0828*	−0.0557	−0.0955*	−0.0591	−0.0609	−0.1076*	−0.1263*	−0.0758*	−0.0931*
	−0.0724*	0.0120	0.0075	−0.0321	0.0279	−0.0483	−0.0257	−0.0693	−0.0467	−0.0422	−0.0717*	−0.0993*	−0.0491	−0.1116*	−0.0423
	−0.0073	−0.0390	−0.0558	0.0091	−0.0539	−0.0771*	0.0037	0.0009	−0.0131	0.0135	−0.0251	−0.0705*	−0.0141	−0.0711*	−0.0592
	−0.0724*	−0.0333	−0.0431	−0.0228	−0.0874*	0.0115	−0.0265	−0.0357	−0.0273	−0.0336	−0.0294	−0.0347	−0.0060	−0.0902*	−0.0046
$\Phi_1 - \Phi_2$	0.0350	−0.0169	−0.0037	−0.2197*	0.0452	0.0934	0.1086	0.0193	−0.0080	0.0254	0.0098	−0.1314	0.0403	0.0986	−0.1050
	0.2004*	0.2663*	0.2561*	0.1761*	0.2277*	0.2151*	0.1704*	0.1461*	0.1615*	0.1022*	0.2072*	0.2786*	0.1808*	0.2070*	0.1746*
	0.1041*	0.1025*	−0.0120	0.1078*	0.0382	0.1021*	0.0855	0.0270	0.0844	0.0184	0.0648	0.0807	0.1610*	0.0078	0.0440
	0.0532	0.0004	0.0206	0.0022	−0.0187	0.0616	0.0216	0.1479*	0.0350	0.0866	0.0605	0.0856	0.0491	0.1032	0.0358
	−0.0025	−0.0158	0.1187*	−0.0233	0.0579	0.0579	0.0910*	0.0526	0.0084	−0.0259	−0.0108	0.0714	0.0186	0.0886*	0.0491
	0.0396	0.0185	−0.0215	−0.0225	0.0430	−0.0940*	−0.0007	0.0087	−0.0038	0.0712	0.0586	−0.0304	−0.0219	0.0506	−0.0499

Table 6.3 Volume-dependent AR coefficients for out-of-sample period

	AXP	BA	BAC	CVX	DD	GE	IBM	JPM	MMM	PG	T	UTX	VZ	WMT	XOM
Φ_2	−0.0621	0.1849	−0.8134	0.1615	0.1245	−0.0961	−0.0122	−0.1536	0.0940	0.0467	0.0280	0.2724	0.0193	0.1682	0.0515
	−0.1493	−0.0195	−0.1469	−0.0320	−0.0388	−0.1902	−0.1089	−0.2338	−0.0741	−0.1588	−0.0695	−0.1672	−0.1202	−0.2645	−0.0546
	0.0232	0.0309	−0.0860	0.0207	−0.0955	−0.0001	−0.0922	−0.0142	−0.0340	−0.1447	−0.1337	−0.0457	−0.0378	−0.1441	0.1019
	0.0765	−0.0348	0.0419	−0.1531	0.0358	−0.0339	−0.1000	−0.0105	−0.0858	−0.0344	−0.0587	−0.1315	−0.1624	−0.1761	−0.0313
	−0.0140	−0.0074	−0.0471	−0.0166	0.0954	0.0080	0.0468	0.0163	0.0035	−0.0086	0.0936	−0.0137	0.0953	−0.0171	−0.0243
	−0.1124	−0.0465	0.0215	−0.0168	0.0262	−0.1164	−0.0901	−0.0116	−0.0449	−0.1511	0.0232	−0.0743	0.0206	0.0060	−0.0288
$\Phi_1 - \Phi_2$	−0.0871	−0.4824	1.4604*	−0.3837	−0.3555	−0.3016	0.0638	0.2203	−0.1985	−0.2112	−0.3132	−0.5388	−0.1803	−0.3516	−0.4225
	0.0505	0.0014	0.2717	−0.1588	−0.0371	0.2046	0.0817	0.1401	−0.0036	0.0584	−0.0537	−0.0103	0.1027	0.0859	−0.3104*
	−0.1546	−0.0504	0.1653	−0.1359	0.0231	−0.0289	0.1082	−0.0705	−0.1392	−0.0700	−0.1390	−0.0665	−0.2730	−0.0354	−0.4349*
	−0.1196	0.2489	−0.1387	0.3498*	0.0440	0.0566	0.2447	−0.0793	0.1787	0.1418	0.1175	0.2812	0.1999	0.2326	0.0489
	0.0903	−0.0327	0.1027	0.0208	−0.0940	0.0435	−0.1235	−0.1021	−0.0261	−0.1228	−0.1415	−0.0248	−0.2768	−0.1355	−0.0046
	0.1282	0.0414	−0.2572	0.0107	−0.0544	0.0295	−0.0058	−0.0826	0.0334	0.2064	0.0180	0.0295	−0.0460	−0.1522	0.0163

independently of each other. The buy/sell trade signals were generated by combining the individual outputs of the two systems. If both outputs have the same sign, then the GVRRL follows that prediction. If they differ in sign, then the GVRRL system compares the strength of each signal (recall that the output is a continuous value in the range $[-1,1]$) and goes with the one having the higher magnitude. In the rare event that the systems output signals having the same magnitude but opposite sign, then the GVRRL system ignores both trade recommendations and holds its previous position.

For the RSRRL models, different values for the smoothness parameter γ, from the set $\{1,5,10,\infty\}$, were considered. Thus, for each dataset, there were 17 types of RRL/RSRRL trading systems. For each one, 100000 different sets of initial network weights were used for training. The mean Sharpe ratios achieved by the top 100 in-sample performers (99.9^{th} quantile) are reported in Table 6.4. The figures in bold correspond to the best performer (excluding the RND traders) for the dataset in question, while the underlined values indicate the best performers among the 4 sub-types for the GRRL and VRRL systems. The results indicate that the RSRRL models outperform the plain RRL model in nearly all cases. The GRRL appears to be underperforming with regards to the other RSRRL systems. More significantly, if the RRL/RSRRL models are compared to the RND system, it can be seen that for the dataset VZ, the random approach yields the best results based on the performance of the top 100 performers. RND also outperforms the plain RRL system for a few other datasets as can be seen from the table. Of course, the reported figures do not give the full picture since the RRL/RSRRL models have a much smaller variance (for all 100000 simulations) and would appear to do significantly better if the quantiles for the whole bunch of traders were to be compared.

The out-of-sample trades were generated by the pooling process described earlier. Table 6.5 summarises the results. The best performers in the test period are in boldface, while the out-of-sample Sharpe ratios achieved by the best in-sample performers are marked by the † symbol. The values for the RND traders correspond to the median Sharpe ratio achieved out of a pool of 100000 candidates for each dataset. The CRRL, short for combi-RRL, is an out-of-sample trader that bases its actions on the combined outputs of the plain RRL and the best in-sample GRRL and VRRL traders (underlined in Table 6.4). After pooling the signals from the respective 'elitist' traders, the CRRL gets a trade recommendation from each type of trader (3 in total in our case) and determines the final trade signal based on a majority rule. In real world trading, a decision has to be made after training about which system to choose for the test period. One could go for the best in-sample performer, but reliance on a single type of trader can have dramatic consequences if it happens to have very poor generalisation capabilities. Hence, the motivation for some combination procedure[2].

The results show that the plain RRL and GRRL traders, who could not match the performance of the other trading systems during the training phase, fare somewhat better in the out-of-sample period. In addition, and more worryingly, there are quite a few traders who end up making losses. Even some of the best-in-sample performers yield negative Sharpe ratios in the test phase, worse than the median random trader. The good in-sample performance is not translated to the out-of-sample period. A Spearman's rank correlation test on a strategy by strategy basis confirms this (none of the coefficients obtained is significant). On a dataset by dataset basis however, some of the results do show statistical significance (5 %), as reported

[2] This could backfire if two or all 3 systems have poor generalisation capabilities, or break down simultaneously. One could incorporate some trailing stop-loss mechanism for each type of trader in the CRRL that could give an indication of recent performance levels, and have some rule to decide upon which trade recommendations to include for determining the final output signal.

Table 6.4 In-sample Sharpe ratios (mean of 100 'elitists')

	AXP	BA	BAC	CVX	DD	GE	IBM	JPM	MMM	PG	T	UTX	VZ	WMT	XOM
RRL	0.0941	0.0894	0.0612	0.0833	0.0644	0.0854	0.0496	0.0613	0.0631	0.0709	0.0507	0.0776	0.0567	0.0799	0.0940
GRRL	0.1044	0.0969	0.0652	0.0868	0.0685	0.0884	0.0668	0.0672	0.0663	0.0758	0.0682	0.0961	0.0569	0.0887	0.0944
	0.0988	0.0873	0.0718	0.0921	0.0843	0.0885	0.0684	0.0702	0.0768	0.0868	0.0742	0.0978	0.0628	0.0981	0.0967
	0.0978	0.0896	0.0718	0.0852	0.0835	0.0909	0.0680	0.0727	0.0826	0.0842	0.0709	0.0955	0.0698	0.0995	0.0963
	0.0953	0.0915	0.0704	0.0912	0.0787	0.0876	0.0655	0.0753	0.0853	0.0821	0.0687	0.0977	**0.0709**	0.0954	0.0916
VRRL	0.1069	0.0875	0.0791	0.0835	0.0814	0.0963	0.0567	0.0713	0.0736	0.0758	0.0676	0.0874	0.0652	0.1044	0.0906
	0.1115	0.0906	0.0942	0.0866	0.0841	0.0948	0.0668	0.0813	0.0686	0.0778	0.0832	0.0927	0.0637	0.1151	0.1022
	0.1103	0.0923	0.0986	0.0819	0.0889	0.0923	0.0743	0.0861	0.0769	0.0843	0.0859	0.0883	0.0667	**0.1187**	**0.1047**
	0.1071	0.0970	0.0932	0.0799	0.0892	0.0945	0.0769	**0.0885**	**0.0968**	0.0866	0.0954	0.0912	0.0698	0.1151	0.0991
LRRL	**0.1134**	0.0877	0.0897	0.0827	0.0697	0.1037	0.0665	0.0688	0.0690	0.0741	0.0808	0.0865	0.0672	0.1022	0.0902
	0.1099	0.0910	0.1001	0.0793	0.0808	0.0903	0.0761	0.0800	0.0738	**0.0949**	0.0936	0.0841	0.0663	0.1017	0.0970
	0.1075	0.0930	0.1020	0.0814	0.0832	0.0894	0.0776	0.0836	0.0841	0.0945	**0.0965**	0.0866	0.0693	0.1022	0.1020
	0.1099	0.0920	**0.1031**	0.0803	0.0848	0.0942	**0.0781**	0.0829	0.0911	0.0864	0.0847	**0.1085**	0.0687	0.1094	0.1012
GVRRL	0.1060	0.0937	0.0750	0.0889	0.0734	0.0918	0.0642	0.0686	0.0717	0.0723	0.0754	0.0877	0.0602	0.0944	0.0898
	0.1042	0.0968	0.0812	0.1052	0.0947	0.1057	0.0712	0.0741	0.0740	0.0828	0.0903	0.0930	0.0602	0.1002	0.0961
	0.1056	0.0960	0.0878	0.1030	**0.0989**	**0.1074**	0.0723	0.0766	0.0801	0.0826	0.0941	0.0915	0.0640	0.1025	0.0993
	0.1056	**0.0977**	0.0811	**0.1068**	0.0948	0.1017	0.0707	0.0885	0.0921	0.0887	0.0884	0.0999	0.0659	0.1075	0.0958
RND	0.0777	0.0779	0.0788	0.0785	0.0770	0.0776	0.0752	0.0760	0.0781	0.0773	0.0775	0.0781	0.0758	0.0780	0.0776

For each type of RSRRL trader, there are 4 sub-types per dataset, corresponding to different values for the smoothness parameter; $\gamma = 1, 5, 10$ and ∞ respectively. The underlined values correspond to the RSRRL system chosen for the out-of-sample combi-trader (CRRL). The figures in boldface represent the best performer among the RRL/RSRRL systems for each dataset.

Table 6.5 Out-of-sample Sharpe ratios

	AXP	BA	BAC	CVX	DD	GE	IBM	JPM	MMM	PG	T	UTX	VZ	WMT	XOM
RRL	0.1012	0.0324	0.0198	0.0439	0.0212	−0.0119	0.0620	0.0551	0.0739	**0.1220**	0.0472	0.0896	0.0759	0.0751	0.0525
GRRL	0.0553	0.0468	−0.0008	0.0484	0.0731	−0.0346	0.0503	0.0140	0.1099	0.1062	0.0974	0.0928	0.0741	0.0274	0.1155
	0.0717	0.0445	−0.0041	**0.0906**	0.0561	0.0231	0.0053	-0.0393	0.0608	0.0790	**0.1178**	0.0910	0.1153	0.0588	0.1006
	0.0437	-0.0043	−0.0151	0.0458	0.0558	**0.0513**	0.0057	0.0064	0.0566	0.0934	0.0873	0.0976	**0.1507**	0.0437	0.1116
	0.0703	0.0090	−0.0114	0.0858	0.0554	0.0198	0.0347	-0.0516	0.0811	0.1004	0.0879	0.0672	0.1393†	0.0366	0.1065
VRRL	**0.1227**	0.0663	0.0384	0.0307	0.0496	−0.0097	0.0931	−0.0613	0.0661	0.0825	0.0114	0.1453	0.0701	0.0464	0.0759
	0.0949	0.0617	0.0437	0.0743	0.0188	−0.0257	0.0136	0.0269	0.0584	0.1091	−0.0097	**0.1531**	0.0664	0.0661	0.0836
	0.1024	0.0562	**0.0554**	0.0528	0.0285	0.0131	−0.0041	0.0149	0.0940	0.0741	−0.0121	0.0937	0.0901	0.0736†	0.0540†
	0.0657	0.0346	0.0511	0.0496	**0.1089**	−0.0455	−0.0133	−0.0635†	0.0494†	0.0624	−0.0043	0.1016	0.0498	0.0623	**0.1298**
LRRL	0.1219†	0.0715	0.0421	0.0159	0.0306	−0.0382	0.0616	0.0052	**0.1546**	0.0893	−0.0007	0.0762	0.0394	0.0641	0.0377
	0.1040	0.0676	0.0391	0.0557	0.0378	−0.0426	−0.0243	−0.0063	0.0989	0.0594†	0.0079	0.0910	0.0278	0.0744	0.0627
	0.0917	0.0380	0.0474	0.0707	0.0228	−0.0479	−0.0344	**0.0617**	0.0311	0.0702	−0.0015†	0.0846	0.0328	0.0850	0.0552
	0.0968	0.0298	0.0497†	0.0511	0.0719	0.0252	0.0516†	−0.0023	0.0291	0.0327	−0.0471	0.0076†	0.0328	0.0628	0.0841
GVRRL	0.0959	0.0511	0.0076	0.0320	0.0670	−0.0241	0.0836	0.0116	0.1155	0.1111	0.0717	0.0746	0.0597	0.0438	0.0653
	0.0927	**0.0797**	0.0152	0.0448	0.0096	−0.0241	**0.1097**	0.0283	0.0938	0.0814	0.0891	0.0749	0.1036	**0.0900**	0.0812
	0.0827	0.0283	0.0478	0.0192	0.0418†	−0.0304†	−0.0235	0.0369	0.1094	0.1064	0.0715	0.0805	0.1276	0.0798	0.0673
	0.1133	0.0101†	0.0304	0.0679†	0.0374	0.0037	0.0256	0.0509	0.0252	0.0273	0.0810	0.0670	0.1101	0.0691	0.1050
CRRL	0.1061	0.0245	−0.0047	0.0782	0.0755	0.0091	0.0356	0.0343	0.1040	0.1087	0.0627	0.0991	0.0925	0.0669	0.0499
RND	0.0202	0.0110	0.0272	0.0174	0.0117	0.0099	0.0098	0.0329	0.0090	0.0084	0.0098	0.0105	0.0115	0.0105	0.0147

For each type of RSRRL trader, there are 4 sub-types per dataset, corresponding to $\gamma = 1, 5, 10$ and ∞ respectively. The values in boldface represent the best out-of-sample performer for each dataset. The values marked by the † symbol correspond to the best in-sample performers.

in Table 6.6, with BAC showing a strong positive correlation between the in-sample and out-of-sample performance. But it is the exception, not the norm, as the figures confirm. Another notable point concerns the performance of the CRRL. As expected, it performs decently but without being spectacular. It struggles with certain stocks, and even leads to a loss with BA, but it appears to be a safer bet than relying on the best in-sample performer to do well out-of-sample. The same could be said about the GVRRL traders. Although they achieve positive Sharpe ratios in the majority of cases, they are no longer among the best performers during the test period. The aggregated approach used in this study adds some risk management in the system, but it is completely decoupled from the learning process, i.e., the trade signal output by the system is not directly involved in the optimisation of the DSR, since the subsystems are independent of each other[3].

Table 6.6 Spearman's Rank Correlation Coefficient (by dataset)

AXP	0.5000*	BA	−0.1863	BAC	0.8186*
CVX	0.0025	DD	−0.0711	GE	−0.1422
IBM	−0.6446*	JPM	0.0588	MMM	−0.6250*
PG	−0.8407*	T	−0.3946	UTX	−0.3137
VZ	−0.0784	WMT	0.2868	XOM	0.0245

6.5.4 Analysis

This section provides a more in-depth discussion about the in-sample and out-of-sample performances of the RSRRL traders, more specifically the VRRL, LRRL and GRRL traders. In any machine learning task, the aim is to have a system that can generalise properly out-of-sample, i.e. good in-sample performance should be translated to the test period. The results have shown that this not necessarily the case for the mechanical traders used in this study, especially for the VRRL and LRRL systems. This immediately brings up the possibility of data-snooping in the form of overfitting. While it cannot be denied that overfitting issues have an undermining role to play in the performance of the traders, it can be argued that, based on the results with the artificial datasets and the analysis tools used, the RRL/RSRRL systems are able to uncover structure in the data, and that the adaptive learning mechanism operates properly.

The in-sample performance showed that the RSRRL systems, especially the VRRL and LRRL, generally performed better than the plain RRL or the RND traders. Based on the regime-dependent AR coefficients (see (6.20)) reported earlier, it can be said that the corresponding transition variables, i.e. volume and the rate of information arrival, are able to partition the input space, such that one regime is predominantly concerned with negative serial correlation while the other typically has slight positive autocorrelation levels. Thus, the subnetworks of these systems are more adapted for certain types of data patterns than others; there is a higher degree of specificity attached to them. The lack of significance of the GARCH-based AR coefficients for the training sets suggest that the volatility indicator is unable to correctly identify the regimes present in the data. This could be due to an incorrect

[3] The 'coupling' could easily be achieved, by feeding the lagged overall trade signals back to each subsystem, in addition to (or instead of) the lagged trade recommendations of each subsystem.

selection of threshold, or simply because there is little or no relationship between volatility level and serial correlation in returns. In any case, this suggests that, at the end of training, the GRRL subnetworks are less specialised than the subnetworks of the VRRL systems for instance. The GRRL situation is a bit similar to the third scenario in the artificial data experiments, in which incorrect regime information is being fed to the system. Whereas for the VRRL and LRRL systems, the situation is analogous, at least to some extent, to the second scenario. But unlike that scenario where it was clear that the superior in-sample performance of the RSRRL was solely due to its learning capabilities, the same claim cannot be made for the real-world datasets, which are typically extremely noisy and the autocorrelation levels are usually low and have a time-varying nature, thereby increasing the risk of overfitting.

For the out-of-sample period, the vast majority of the regime-dependent AR coefficients were found to be statistically insignificant. This tends to suggest that there is a change in market conditions during the test period; this could explain the lack of correlation between the in-sample and out-of-sample performance. Recall that the test period includes the financial crisis, and thus, for most of the datasets, the out-of-sample regimes are very different from the in-sample regimes. Figure 6.4 shows the cumulative wealth profiles (in %) pertaining to the RRL and the RSRRL systems with $\gamma = 5$ for each dataset. It can be seen that in the initial stages of the test period, for many of the stocks under consideration, the wealth profiles of the different traders are rather similar. But around the Sep/Oct 2008 period, at the height of the crisis, most of the traders witness a shock in their performance; some of the systems thrive in the immediate aftermath, while others break down. In the latter stages of the test period though, the change in performance levels is more gradual. The graphs convey the impression that the Sep/Oct 2008 period effectively has a big say in determining the overall out-of-sample performance. The emergence of altogether new regimes, or at least regimes that have little correlation with their in-sample equivalent, have an impact on overall performance. The LRRL systems appear to be hit the hardest. This could be explained by the fact that their subnetworks undergo a relatively high level of specialisation during the training phase and are therefore more prone to being adversely affected by regime changes. From Figures 6.2 and 6.3, it can be seen that there is a significant increase in the raw trading volume in the whereabouts of the beginning of the test period. This is true for many of the other stocks. The new volume dynamics might have completely different price-volume characteristics, and thus the the LRRL sysems invariably suffer out-of-sample. The MDH model obtained from the training data would be inappropriate for the test period, meaning that the estimates for the rate of information arrival during the test period are no longer trustworthy. The GRRL systems on the other hand, have less specialised subnetworks than the LRRL systems. During the test phase, they can more easily adapt to the changes in market conditions. Moreover, since the GRRL, by virtue of the threshold selection, is predominantly in a high-volatility regime during the test period (see e.g., the volatility profile in Figure 6.3), the weight updates are mostly focused on the subnetwork attributed to that regime. The GRRL effectively becomes like the standard RRL. The VRRL and LRRL systems, on the other hand, constantly undergo regime shifts in the out-of-sample phase, and have more difficulty in readjusting the weights of their subnetworks to adapt to the changing conditions.

The main concern regarding the results is that the out-of-sample performance is not reflective of the in-sample performance. While overfitting is probably an issue, we argue that the lack of persistence in the results is mostly due to the drastic changes in market conditions during the test period. The out-of-sample equity curves tend to support this, as well as the control experiments with the artificial datasets. There are a couple of instances (see e.g., the wealth profile for 'GE') where the adaptive learning capabilities of the systems can be questioned, but by and large, the systems recover even after going through extremely turbulent periods.

Had overfitting been the overriding factor during the training phase, the out-of-sample performance of these traders would have been more like that of a random trader; this is definitely not the case for the majority of the datasets.

Table 6.7 Out-of-sample Results for FTSE and NASDAQ datasets

Dataset	Type	Period 1	Period 2	Period 3	All Periods
FTSE	GRRL	50.9 % (0.0082)	61.8 % (0.0178*)	56.4 % (0.0073)	56.4 % (0.0111*)
	VRRL	54.5 % (0.0100)	63.6 % (0.0201*)	58.1 % (0.0042)	58.8 % (0.0115*)
NASDAQ	GRRL	47.2 % (0.0017)	43.8 % (−0.0075)	46.1 % (−0.0051)	45.7 % (−0.0037)
	VRRL	52.8 % (0.0013)	48.3 % (−0.0088)	52.8 % (0.0026)	51.3 % (−0.0016)

The values correspond to the proportion of cases where the RSRRL system in question outperforms the RRL system. The values enclosed in brackets refer to the mean difference the RSRRL and RRL Sharpe ratios ($RSRRL - RRL$)

6.5.5 Further Experiments

Additional experiments were carried out in an attempt to see more clearly about the efficacy of the RSRRL systems relative to the RRL traders. Only the VRRL and GRRL systems were considered, mainly because it is computationally cheap to get the desired transition values, as opposed to the LRRL system which involves high computational costs. Two datasets were considered, the FTSE dataset which consisted of 55 stocks from the FTSE 100 index, and the NASDAQ dataset, which encompassed 89 stocks from the NASDAQ-100 index. The datasets spanned the period from August 2003 upto August 2010. A rolling window approach was used, with the training period consisting of 4 years of data (1000 trading days) and the test period being 1-year long (250 trading days). This effectively amounted to 3 out-of-sample periods, the first one starting in August 2007 and ending in August 2008. The experimental setup was as described earlier, except that only 5000 different sets of initial network weights were used for training, and the top 50 in-sample performers were selected for the pooling process. Additionally, only the threshold version of the RSRRL was investigated. The out-of-sample results are reported in Table 6.7. The % values correspond to the proportion of cases in which the RSRRL system in question achieves a higher Sharpe ratio than the standard RRL, whereas the figures enclosed in brackets correspond to the mean difference in Sharpe ratios. Statistically significant (5%) results are marked with the * symbol (alternative hypothesis of $RSRRL > RRL$). For instance, for the FTSE dataset, the GRRL performs better than the RRL in 51.9 % of cases for the first out-of-sample period, and the mean difference between the two types of traders is 0.0082 in favour of the GRRL (results not statistically significant). It can be seen that the RSRRL traders consistently outperform the RRL traders for the FTSE stocks over the different periods, but cannot repeat this feat when subjected to the NASDAQ dataset. For the FTSE dataset, the proportion of cases in which positive Sharpe ratios are achieved is 61.2%, 69.7%, and 73.3% for the RRL, GRRL and VRRL systems respectively. For the

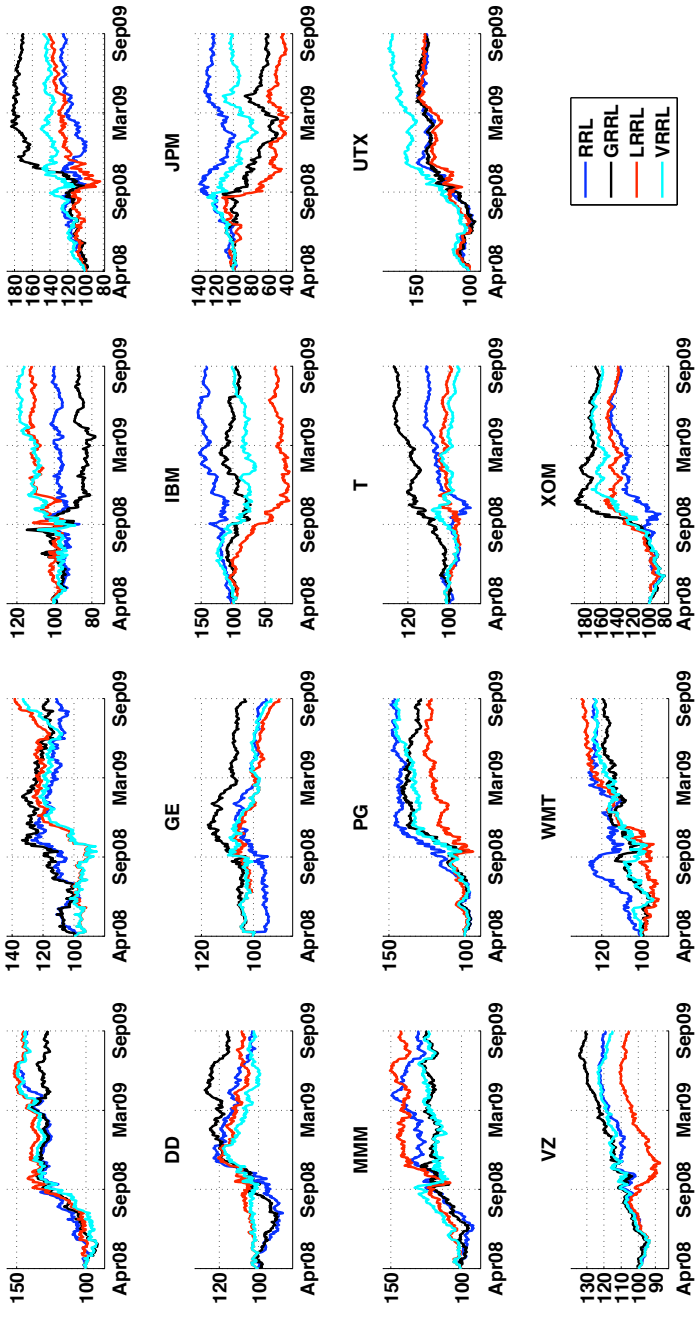


Fig. 6.4 The out-of-sample cumulative wealth profiles (in %) for the RRL and RSRRL traders with $\gamma = 5$.

NASDAQ dataset, the values are 71.5/65.2/68.9. These experiments tend to confirm that the RRL and RSRRL systems are able to find structure in real datasets and can lead to profitable situations. As far as the performance of the RSRRL relative to the RRL with regards to real datasets is concerned, there is not enough evidence to suggest that the former is a better alternative than the latter. For the RSRRL to be consistently superior, certain conditions should hold, as the artificial experiments showed. With real datasets, which are typically very noisy and exhibit time-varying autocorrelation levels, the higher complexity of the RSRRL could make it more prone to overfitting and therefore less robust to out-of-sample generalisation.

6.6 Conclusion

In this study, we investigated the regime-switching recurrent reinforcement learning (RSRRL) model, first proposed in Maringer and Ramtohul (2010), and which is an extended version of the recurrent reinforcement learning (RRL) algorithm put forward by [22]. We looked at different variants of the RSRRL, with different types of transition variables, and compared their performance with the basic RRL model in automated trading experiments. We used both artificial data and real-world data for our comparisons. We also emphasised on the importance of correct identification of the regimes, and advocated for the use of volatility, trading volume, and the rate of information arrival as transition variables, based on past empirical evidence and. We used a GARCH(1,1) process to model volatility, the Mixture Distribution Hypothesis model for estimating the rate of information arrival, and a preprocessed form of volume for the purpose of the RSRRL systems.

Based on results with artificial data, we found that, in general, the performance of the RSRRL matches that of the RRL in datasets having a single regime or regimes that are closely related to each other. However, the RSRRL significantly outperforms the RRL in situations where the datasets are characterised by distinctly different regimes, provided that the regimes are correctly identified and there is some correlation between the in- and out-of-sample regimes.

The in-sample results with the real-world datasets provided enough evidence to justify the integration of the RRL in a regime-switching framework and validated the choice of transition variables. The good performance in-sample was however not repeated out-of-sample, and many of the RSRRL systems could not match the performance of the standard RRL traders. We showed that this was most probably due to a drastic change in market conditions during the test period, which coincided with the financial crisis. Further experiments were conducted in a bid to investigate the out-of-sample performances of the two types of traders, and it was found that the RSRRL systems yield superior results with FTSE stocks but are outperformed in some periods by the standard RRL with the NASDAQ stocks. The results from both the artificial and real-world datasets suggest that the RSRRL can only consistently outperform the RRL when certain criteria are met, but generally does not have the robustness levels of the latter.

Of the three transition variables studied, there is no clear-cut winner if both the in-sample and out-of-sample results were to be considered. The GARCH-based models are not able to correctly identify the regimes; they might benefit from the location parameter c being chosen using some sophisticated approach, or alternatively, having a time-varying threshold for better regime identification. The VRRL systems, although far from being perfect, do achieve a certain level of success, doing as well as, if not better than, the LRRL systems. The latter seem to suffer the most with regards to the drastic change in economic conditions. Because of the fragility associated with these systems, as well as the relatively tedious process of

extracting the latent variables, it seems more reasonable to focus on the VRRL rather than the LRRL. Additionally, results with the NASDAQ and FTSE datasets seem to suggest that volume is a better indicator than GARCH-based volatility. In future, work could be focused on volume-based information, e.g., abnormal trading volume or/and the associated trading volume in futures markets.

In this study, the model parameters such as the learning rate ρ and the adaptation rate η have been chosen arbitrarily. Consider for instance the GRRL system. The high volatility regime is typically associated with relatively high returns, and therefore would require parameters that are different from the network associated with the other regime (especially for the threshold model). In addition, the transition function parameters γ and c could also be chosen in a more sophisticated manner. Future work could include an optimisation routine that aims at finding the correct network and transition function parameters in addition to the DSR maximisation, with due care taken to minimise the effects of overfitting.

The flexibility of the RSRRL model means that it can be easily modified to suit the financial problem being investigated. The model can be customised to match the needs of the problem and transition variables combined in a variety of ways to best match the features of the application environment and the beliefs of the investor. It can easily be extended to accommodate more regimes and/or more indicator variables. A brief glimpse of this flexibility has been given in this study with regards to the GVRRL and CRRL traders, who perform reasonably well, and could potentially do even better, especially out-of-sample, if a certain degree of coupling is introduced within each system.

A Estimation of Transition Variables

A.1 Volatility

A simple GARCH(1,1) model, defined below, was chosen to represent the daily volatility process. The choice was motivated by the parsimony and the widely-acknowledged reliability of this type of model. For each dataset, the respective GARCH parameters were estimated over the training data, and subsequently used to make one-day-ahead forecasts during the test period. The parameter estimates are summarised in Table 6.8.

$$\begin{aligned} r_t &= C + \varepsilon_t, \qquad \text{with} \qquad \varepsilon_t \sim N(0, h_t) \\ h_t &= K + \alpha \varepsilon_{t-1}^2 + \beta h_{t-1}. \end{aligned}$$

The simplified form of the log likelihood function for this model is

$$\sum_{t=1}^{T} \left(-\log(h_t) - \frac{\varepsilon_t^2}{h_t} \right).$$

A.2 Volume

For the volume-based RSRRL model, a preprocessed version of the raw volume series was used, as described in [1]. The raw volume series was divided by a stochastic trend component obtained using kernel regression, in an attempt to make the series stationary. The raw volume values are divided by the non-linear trend component, which corresponds to kernel estimates, to yield the desired volume series.

Table 6.8 GARCH(1,1) parameter estimates

	AXP	BA	BAC	CVX	DD	GE	IBM	JPM	MMM	PG	T	UTX	VZ	WMT	XOM
$C^\star$	0.7103	1.1056	0.3093	0.7652	0.3043	0.3082	0.5075	0.4620	0.4895	0.4718	0.4413	1.2056	0.1168	-0.0203	0.6909
$K^\star$	1.9036	8.7194	0.9640	6.3813	1.3769	0.2921	4.0950	1.0441	12.9465	0.7159	0.4094	4.7740	1.3275	0.8633	3.9896
α	0.0857	0.1007	0.0410	0.0685	0.0387	0.0310	0.1034	0.0660	0.0775	0.0227	0.0339	0.1260	0.0550	0.0299	0.0594
β	0.9138	0.8776	0.9561	0.9000	0.9565	0.9684	0.8870	0.9340	0.8616	0.9713	0.9653	0.8740	0.9411	0.9659	0.9219

* The tabulated values for C and K have been multiplied by 10^3 and 10^6 respectively for keeping the precision while maintaining compactness and readability; e.g, the actual value of K for the AXP dataset is 1.9036×10^{-6}.

Table 6.9 MDH parameter estimates

	AXP	BA	BAC	CVX	DD	GE	IBM	JPM	MMM	PG	T	UTX	VZ	WMT	XOM
σ_r	1.7794	1.6491	1.4525	1.3176	1.4152	1.4822	1.4170	1.7832	1.3055	1.0510	1.6169	1.5064	1.5859	1.4257	1.3029
μ_0	0.3563	0.3810	0.2934	0.4223	0.4863	0.4434	0.6100	0.4441	0.4525	0.5254	0.4776	0.4421	0.4870	0.5331	0.6253
μ_w	0.5115	0.5118	0.5424	0.5023	0.4206	0.4235	0.2746	0.4128	0.5107	0.3660	0.4156	0.4517	0.4594	0.3644	0.3001
σ_v	0.1631	0.1574	0.1323	0.1537	0.1735	0.1462	0.1910	0.1661	0.1932	0.1755	0.1575	0.1745	0.1717	0.1673	0.1574
δ_λ	0.7283	0.6332	0.8047	0.7029	0.6706	0.7545	0.7445	0.7719	0.7139	0.6912	0.7004	0.7114	0.7190	0.6926	0.7207
ν_λ	0.3901	0.4523	0.3193	0.3218	0.4478	0.3920	0.5389	0.4385	0.4843	0.4753	0.4518	0.4098	0.4319	0.4744	0.4153

A.3 Rate of Information Arrival

The MDH model by [1] was used for obtaining estimates of the rate of information arrival. The first stage of the estimation consisted of finding the model parameters by maximum likelihood. The problem could be formulated as follows. Let $Y_T = \{y_t\}_{t=1}^T$ denote the matrix of observable variables with $y_t = (r_t, v_t)$ and $\Lambda_T = \{\lambda_t\}_{t=1}^T$ the matrix of latent variable. The marginal likelihood function can be obtained by integrating out Λ_T from the joint probability function of Y_T and Λ_T,

$$L(\theta; Y_T) = \int f(Y_T, X_T|\theta) dX_T = \int g(Y_T|\Lambda_T, \theta) h(\Lambda_t|\theta) d\Lambda_T,$$

$$g(Y_T|\Lambda_T, \theta) = \prod_{t=1}^{T} g_t(y_t|\lambda_t, \theta),$$

$$h(\Lambda_t|\theta) = \prod_{t=1}^{T} h_t(\lambda_t|\lambda_{t-1}, r_{t-1}, \theta),$$

where θ denotes the vector of parameters to be estimated. The likelihood function is therefore a multiple integral over T dimensions. $g_t(y_t|x_t, \theta)$ denotes the joint density of observable return and volume $y_t = (r_t, v_t)$ conditional on the latent variables λ_t. The process $\{r_t\}$ consisted of the daily log returns (in %) , while $\{v_t\}$ was obtained by preprocessing the raw volume series using the aforementioned detrending technique.

An approach proposed by [4], known as the Gaussian-EIS Particle Filter was used for computing the likelihoods. It is based on the accelerated Gaussian importance sampling (AGIS) procedure developed by [3] and further refined by [24]. See for instance [18] of how the original approach could be used for the estimation of MDH model parameters, [25] for other application areas, and [6] for an excellent tutorial on particle filtering. The approach proposed by [4] incorporates the EIS algorithm in a particle filter framework that enables computation of the likelihood function and estimation of the latent variables in the same run.

The model parameters obtained through this technique are summarised in Table 6.9. It is worth mentioning that the persistence parameter δ_λ is less than the persistence parameter β from the GARCH(1,1) model, which highlights the inability of the MDH to fully account for the ARCH effects in stock price series, as reported by [1] and [18]. Once the model parameters had been determined, the filtered values $\mathrm{E}[\exp(\lambda_t)|r_t, v_t)]$ were extracted, which correspond to the daily number of information arrivals I_t. These estimates were used as the transition variable.

References

1. Andersen, T.G.: Return volatility and trading volume: An information flow interpretation of stochastic volatility. Journal of Finance 51(1), 169–204 (1996)
2. Bertoluzzo, F., Corazza, M.: Making financial trading by recurrent reinforcement learning. In: Apolloni, B., Howlett, R.J., Jain, L. (eds.) KES 2007, Part II. LNCS (LNAI), vol. 4693, pp. 619–626. Springer, Heidelberg (2007), http://dx.doi.org/10.1007/978-3-540-74827-4_78
3. Danielsson, J., Richard, J.F.: Accelerated gaussian importance sampler with application to dynamic latent variable models. Journal of Applied Econometrics 8, 153–173 (1993)

4. DeJong, D.N., Dharmarajan, H., Liesenfeld, R., Richard, J.F.: An efficient filtering approach to likelihood approximation for state-space representations. Economics Working Papers, 25. Christian-Albrechts-University of Kiel, Department of Economics (2007), http://ideas.repec.org/p/zbw/cauewp/6339.html
5. Dempster, M.A.H., Leemans, V.: An automated fx trading system using adaptive reinforcement learning. Expert Systems with Applications 30(3), 543–552 (2006); Intelligent Information Systems for Financial Engineering
6. Doucet, A., Johansen, A.M.: A tutorial on particle filtering and smoothing: Fifteen years later. The Oxford Handbook of Nonlinear Filtering (2009)
7. Epps, T.W.: Security price changes and transaction volumes: Theory and evidence. The American Economic Review 65(4), 586–597 (1975)
8. Epps, T.W.: Security price changes and transaction volumes: Some additional evidence. Journal of Financial and Quantitative Analysis 12(1), 141–146 (1977)
9. Fleming, J., Kirby, C., Ostdiek, B.: Stochastic volatility, trading volume, and the daily flow of information. Journal of Business 79(3), 1551–1590 (2006)
10. Franses, P.H., van Dijk, D.: Nonlinear time series models in empirical finance. Cambridge University Press, Cambridge (2000)
11. Gold, C.: FX trading via recurrent reinforcement learning. In: Proceedings of IEEE International Conference on Computational Intelligence for Financial Engineering, pp. 363–370 (2003), doi:10.1109/CIFER.2003.1196283
12. Hamilton, J.D.: A new approach to the economic analysis of nonstationary time series and the business cycle. Econometrica 57(2), 357–384 (1989)
13. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. Journal of Artificial Intelligence 4(1), 237–285 (1996)
14. Karpoff, J.M.: The relation between price changes and trading volume: A survey. The Journal of Financial and Quantitative Analysis 22(1), 109–126 (1987)
15. Koutmos, G.: Feedback trading and the autocorrelation pattern of stock returns: Further empirical evidence. Journal of International Money and Finance 16(4), 625–636 (1997), doi:10.1016/S0261-5606(97)00021-1
16. Lamoureux, C.G., Lastrapes, W.D.: Endogenous trading volume and momentum in stock-return volatility. Journal of Business and Economic Statistics 12(2), 253–260 (1994)
17. LeBaron, B.: Some relations between volatility and serial correlations in stock market returns. Journal of Business 65(2), 199–219 (1992)
18. Liesenfeld, R.: A generalized bivariate mixture model for stock price volatility and trading volume. Journal of Econometrics 104(1), 141–178 (2001)
19. Maringer, D. G., Ramtohul, T.: Threshold recurrent reinforcement learning model for automated trading. In: Di Chio, C., Brabazon, A., Di Caro, G.A., Ebner, M., Farooq, M., Fink, A., Grahl, J., Greenfield, G., Machado, P., O'Neill, M., Tarantino, E., Urquhart, N. (eds.) EvoApplications 2010. LNCS, vol. 6025, pp. 212–221. Springer, Heidelberg (2010)
20. McKenzie, M.D., Faff, R.W.: The determinants of conditional autocorrelation in stock returns. Journal of Financial Research 26(2), 259–274 (2003)
21. Moody, J., Saffell, M.: Learning to trade via direct reinforcement. IEEE Transactions on Neural Networks 12(4), 875–889 (2001), doi:10.1109/72.935097
22. Moody, J., Wu, L.: Optimization of trading systems and portfolios. In: Proceedings of the IEEE/IAFE 1997, Computational Intelligence for Financial Engineering (CIFEr), pp. 300–307 (1997), doi:10.1109/CIFER.1997.618952
23. Moody, J., Wu, L., Liao, Y., Saffell, M.: Performance functions and reinforcement learning for trading systems and portfolios. Journal of Forecasting 17(56), 441–470 (1998)

24. Richard, J.F.: Effcient high-dimensional monte carlo importance sampling. Tech. rep., University of Pittsburgh (1998)
25. Richard, J.F., Zhang, W.: Efficient high-dimensional importance sampling. Working Papers 321. University of Pittsburgh, Department of Economics (2007), http://ideas.repec.org/p/pit/wpaper/321.html
26. Sentana, E., Wadhwani, S.B.: Feedback traders and stock return autocorrelations: Evidence from a century of daily data. Economic Journal 102(411), 415–425 (1992)
27. Sharpe, W.F.: Mutual fund performance. Journal of Business 39, 119–138 (1966)
28. Sun, W.: Relationship between trading volume and security prices and returns. Tech. rep., Technical Report (2003)
29. Sutton, R.S., Barto, A.G.: Introduction to Reinforcement Learning. MIT Press, Cambridge (1998)
30. Tauchen, G.E., Pitts, M.: The price variability-volume relationship on speculative markets. Econometrica 51(2), 485–505 (1983)
31. Teräsvirta, T.: Specification, estimation, and evaluation of smooth transition autoregressive models. Journal of the American Statistical Association 89(425), 208–218 (1994)
32. Tong, H.: On a threshold model. In: Chen, C. (ed.) Pattern Recognition and Signal Processing, pp. 101–141. Sijthoff & Noordhoff (1978)
33. Werbos, P.J.: Backpropagation through time: What it does and how to do it. Proceedings of the IEEE 78(10), 1550–1560 (1990)
34. White, H.: Some asymptotic results for learning in single hidden-layer feedforward network models. Journal of the American Statistical Association 84(408), 1003–1013 (1989)
35. Wood, R.A., McInish, T.H., Ord, J.K.: An investigation of transactions data for NYSE stocks. The Journal of Finance 40(3), 723–739 (1985)
36. Ying, C.C.: Stock market prices and volumes of sales. Econometrica 34(3), 676–685 (1966)

7

An Evolutionary Algorithmic Investigation of US Corporate Payout Policy Determination

Alexandros Agapitos[1], Abhinav Goyal[2], and Cal Muckley[2]

[1] Natural Computing Research and Applications Group & Financial Mathematics and Computation Research Cluster, Complex and Adaptive Systems Laboratory, University College Dublin, Ireland
alexandros.agapitos@ucd.ie
[2] School of Business, University College Dublin, Ireland
{cal.muckley,abhinav.goyal}@ucd.ie

Summary. This Chapter examines cash dividends and share repurchases in the United States during the period 1990 to 2008. In the extant literature a variety of classical statistical methodologies have been adopted, foremost among these is the method of panel regression modelling. Instead, in this Chapter, we have informed our model specifications and our coefficient estimates using a genetic program. Our model captures effects from a wide range of pertinent proxy variables related to the agency cost-based life cycle theory, the signalling theory and the catering theory of corporate payout policy determination. In line with the extant literature, our findings indicate the predominant importance of the agency-cost based life cycle theory. The adopted evolutionary algorithm approach also provides important new insights concerning the influence of firm size, the concentration of firm ownership and cash flow uncertainty with respect to corporate payout policy determination in the United States.

7.1 Introduction

In this Chapter we examine United States corporate payout policy determination using a genetic programming methodology, during the period 1990 to 2008. The term corporate payout policy relates to the disbursing of cash, by a corporation, to shareholders by way of cash dividends and/or share repurchases. Clearly, alongside investment and capital structure optimisation this is a chief responsibility of an organisation's financial officer.

The adopted Genetic Programming [21] methodology is also known as a symbolic regression methodology. It identifies the functional form as well as the optimal coefficients which optimises a program-performance criterion. As a result, this class of model estimation methodology is complementary to the random effects panel regression methodology, typically adopted in the conventional mainstream literature regarding corporate payout policy determination [14]. In addition, it is worthwhile emphasizing that while genetic programming techniques have been adopted to specify trading rules in foreign exchange markets [27, 1] and more broadly in financial modelling [7], there is no contribution to the extant literature which avails of Genetic Programming techniques to evaluate the determination of corporate payout policy.

A. Brabazon et al. (Eds.): Natural Computing in Comput. Finance, SCI 380, pp. 123–139.
springerlink.com

By way of a foundation, to the topic of corporate payout policy determination, the Miller-Modigliani irrelevance proposition [25] indicates that, within a stylised setting, once corporate investment policy is optimal (i.e. once the Fisherian Net Present Value rule is satisfied), corporate payout policy has no implication for the value of the firm. In this setting, corporate payout policy merely involves different methods of distributing free cash flows - by way of cash dividends or share repurchases - and hence has no implication for the value arising from investment decisions. Notwithstanding this, DeAngelo, and DeAngelo [9] conclude that the distribution/retention decision with regard to free cash flows, even assuming the stylised setting outlined in the Miller-Modigliani proposition [25], has 'first-order value consequences'. In brief, this follows from the fact that the feasible set of distribution/retention decisions, in the Miller-Modigliani stylised setting, is exactly the optimal set, i.e. full payout. Evidently, this precludes a payout policy decision. To mitigate for this oversight, DeAngelo and DeAngelo [9] advocate an extension of the classic Fisherian Net Present Value 'rule' with regard to capital budgeting decisions, to include the distribution of the full present value of free cash flows during the life of the firm. Essentially, it is now evident that there is considerable scope for value creation and destruction, by means of corporate payout policy. As a result, the determination of corporate payout policy merits careful attention.

In relaxing the configuration of assumptions underpinning the Miller-Modigliani proposition extended to include the assumption of full payout, several theories, which are mutually inclusive in principle, arise concerning the determination of the timing and form of optimal corporate payout policy. The open question appears to hinge on the relative importance of these theories with regard to explaining the determination of corporate payout policy. In particular, these theories comprise: *first*, the so-called agency cost-based life cycle theory (see [9, 11, 15, 18] which implies that the decision to distribute or retain free cash flows, a trade-off between the prospect of credit constraints and excessive financial slack, varies according to the evolution of the phases of the firm's life cycle i.e. as typified by a firm's size, profitability, the nature of its capital structure and the growth opportunities of the firm. The reconciling of Jensen's agency cost-based theory [19, 23] with the life-cycle theory appears particularly beneficial. Indeed, the agency requirement of persuasion, on the part of the principal, for the agent to distribute free cash flows may be requisite such that the corporation disgorges cash. *Second*, the so-called signaling theory [5, 20, 26] which emphasises the importance of utilising corporate payout policy, to circumvent the information asymmetry which may arise between the management of the firm, who enjoy insider information, and the firm's investors. *Third*, the catering theory [3, 4] of corporate payout policy determination, which highlights the importance of corporate payout policy to satisfy the preferences of various, possibly time-varying, heterogenous payout clienteles. While there has been considerable evidence gathered to the contrary, with respect to the general increase in the level of dividends and a general tendency to increase dividends [11, 12], it cannot be rejected that this latter theory may contribute, albeit, perhaps, in a secondary manner, to the understanding in the extant literature of corporate payout policy determination.

Our findings may be summarised adopting three main sets of key points. First, we show the best evolved symbolic regression models with respect to the root mean square error criterion, these optimal specifications are evidently different to the conventional random effects panel regression model specifications. According to this approach, the theory which best explicates cash dividend payouts and share repurchases is the life-cycle theory of corporate payout policy determination, and it appears that a hybrid hypothesis with respect to both the agency cost-based theory and the life-cyle theory is of particular interest. Second, adopting the Pearson correlation coefficient, scatter plots and regressor containment we show the nature of the relation between individual regressors and identified optimal model specifications

and expression-trees of the best-of-run individuals. Specifically, in line with the findings in the extant literature as well as the various theories of corporate payout, we document a positive relation between both the size of the firm and the earnings to assets ratio and cash dividend payouts and share repurchases. In the same vein, we document a compelling negative relation between the concentration of firm ownership and corporate payout and a somewhat weaker relation between cash flow uncertainty and corporate payout. These confirmations of earlier published findings are important in light of the relatively flexible model specification adopted in this Chapter. Third, we adopt a specification which comprises all the explanatory proxy variables in a single model specification. Consistent with theory, this auxiliary extended model convincingly out-performs the individual models with regard to the root mean squared error criterion. Once again the aforementioned relations, with regard to the direction of effects, are evident in the data. Taken together, our findings support the agency cost-based life-cycle theory of corporate payout policy determination despite the adoption of a distinctive model specification modelling methodology.

The chapter is organized as follows. In section 7.2, we present the sample of data examined, we outline the proxy variables adopted and their hypothesised relations with respect to the theories of corporate payout policy determination assessed. In section 7.3 we describe the adopted genetic programming methodology and section 7.4 we examine our findings. We offer some concluding remarks in section 7.5.

7.2 Literature Review

Published findings with respect to United States corporate payout policy can be summarised by adopting five key points. *First*, as documented by Fama and French [15], due to equally important factors: changing firm characteristics - low profitability, high growth opportunities and relatively intangible fixed assets - as well as a declining propensity to pay, the fraction of US industrial firms paying cash dividends has dropped considerably from 66.5% in 1978 to 20.8% in 1999. *Second*, following a Securities and Exchange Commissions Ruling in 1982 legalising open market repurchases by corporate management, this tax favoured and flexible method, relative to cash dividends, of disbursing cash to shareholders has become of first order importance. Indeed, Skinner [32] indicates that share repurchases are now the preferred method for distributing cash to investors in the United States. *Third* the total value (nominal and real) of cash dividends and share repurchases has risen almost incessantly for several decades. In fact, Weston and Siu [33] show that the US corporate sector's cash dividend payout ratio has increased from 40% in 1971 to around 60% in 1990 and to 81% in 2001. Once share repurchases are included this payout ratio reaches 116 % in 2001.

As a *fourth* key point, DeAngelo, DeAngelo and Skinner [10] show that there has been increasing levels of concentration of dividends and earnings since the 1980s - nowadays a mere 25 firms account for over 50% of industrial earnings and dividends in the United States. In addition it is indicated that while there has been a decline in the number of industrial payers since 1978, the number of financial and utility payers has increased, as has the total value of their payout. These findings show how a declining tendency to disburse cash dividends, an increasing tendency of cash dividend payers to repurchase shares and a rising total value of real payout are internally consistent. Fifth, DeAngelo, DeAngelo and Stulz [11] show that a firm at an early phase of its financial life-cycle with a corresponding low level of retained earnings to total contributed capital in its equity capitalization will tend not to pay dividends or will pay very little by way of a dividend. In the same vein, mature firms are inclined to pay dividends and to pay relatively more. In addition, DeAngelo, DeAngelo and Stulz emphasise

the agency costs which may arise if more mature firms were not to increasingly disburse cash to shareholders with the decline of their investment opportunity sets in line with the maturation of their financial life-cycles. With this backdrop in mind with regard to corporate payout policy in the United States we turn to the determination of corporate payout policy in the United States.

7.2.1 Sample Data and Proxy Explanatory Valriables

Our data is sourced in the *Worldscope* database as detailed in Table 7.1. The sample extends from 1990 through to 2008 inclusive. The data is tailored such that it excludes firms in the financial and utilities industries, as well as American Depositary Receipts and foreign firms. The sample also excludes firms whose dividends are greater than their total sales, firms whose dividend, net income or sales figures are omitted and firms with negative book value of equity, market to book ratio, sales, dividends or share repurchases. In addition, we search the databases for active as well as dead and suspended listings in order to avoid survivor bias. Otherwise, the sample comprises of every firm headquartered in United States and listed on New York Stock Exchange (NYSE), for which there is available our set of proxy explanatory variables. These filters yield 1665 industrial (and transport) firms. Within this group, 960 firms disclose their cash dividend policy in 1990 of which 588 are cash dividend payers. This figure is 1059 in 2008 with a negligible increase in cash dividend payers to 596. Turning to repurchase observations, there are 958 firms which disclose their share repurchases policy in 1990 with 399 firms observed to conduct share repurchases. This figure grows to 1039 firms disclosing their policy in 2008 with 670 firms conducting share repurchases. The total sample includes 14,507 firm-year observations on cash dividends of which 7,846 are cash dividend payers and 6,661 are firms that do not pay cash dividends. There are 14,405 firm-year observations on share repurchases of which 7,571 firm-year observations are for repurchasers and 6,834 for non-repurchasers. Our study examines the United States (primarily NYSE) circumstances, separately investigating *cash dividend paying* and *share repurchasing* firms, using a real US Dollar numeraire (1990 prices) in each instance.

Our principal payout variables are cash dividends (DIV) and share repurchases (SR). Share repurchases correspond to actual gross amounts. We arrange our principal proxy explanatory variables into groups according to their advocated theoretical linkages with respect to explicating the agency cost-based theory, the catering theory, the life-cycle theory and the signaling theory of corporate payout policy.

We assess the empirical importance of the agency cost-based theory adopting 3 proxy explanatory variables. First, following Dittmar and Mahrt-Smith [13] and Pinkowitz *et al.* [29] we adopt cash and short term investments (CASH) as a measurement of prospective agency costs. The greater these prospective costs, the greater the expected corporate payout. In a similar vein, following Chay and Suh [8] and LaPorta *et al.* [23] the more concentrated the ownership of the firm (OWN), the smaller the scope for prospective agency costs. Finally, in regard to agency costs, following Black [6], Jensen [19] and von Eije and Megginson [14], we adopt a leverage ratio (LR) i.e. the book value of debt divided by the book value of assets, to approximate for the scope for prospective agency costs. The greater the leverage of a firm the smaller the scope for prospective agency costs and the smaller the expected payoff. Alternatively, higher leverage may proxy for a firm's maturity which would imply a possible positive relation between firm payout and the leverage ratio (LR).

With regard to catering theory, we follow Baker and Wurgler [3, 4] and specify a dummy variable (CCD) that takes the value 1 if the natural logarithm of the median market to book value of a paying firm is greater than that of the median non-paying firm, otherwise it takes the

Table 7.1 Description of the variables used in the random effects panel regression models

Regressands	***Description***
Cash Dividends (DIV)	The logarithm of the total real value of common cash dividends distributed by the firm, in United States dollar 1990 prices. The logistic random effects panel regression models are specified to include a dummy variable =1 if cash dividends are paid, otherwise zero.
Share Repurchases (SR)	The logarithm of the total real value of open market share repurchases undertaken by the firm, in United States dollar 1990 prices. The logistic random effects panel regression models are specified to include a dummy variable =1, if share repurchases occur, otherwise zero.
Regressors	***Description***
Firm Ownership (OWN)	The percentage of common stock held by the ten largest shareholders.
Cash Holding (CASH)	The sum of cash and short term investments as a percentage of the total assets of the firm.
Leverage Rate (LR)	The sum of short-term and long-term debt as a percentage of total assets.
R & D Exp. (RnD)	Research and development expenses percentage of the total assets of the firm.
Retained Earnings (RETE)	The retained earnings as a percentage of the market value of firm equity.
Asset Growth (DAA)	The relative (percentage) change in the real value of total assets.
Market to Book Value (MBF)	The market to book value of the firm.
Market Value (SIZE)	Percentile ranking (annual) of a firm with respect to the criterion of market value.
Capital Expenses (CapEx)	It represent the funds used to acquire fixed assets other than those associated with acquisitions percentage of total assets.
Stock Return (DPP)	The annual percentage change in stock price measured at the end of the previous year.
Earnings Ratio (EA)	The firm earnings before interest but after tax as a percentage of total assets.
Catering Theory Proxy Variable (CCD)	A dummy variable (annual), which indicates whether the cash dividend payer (share repurchaser) has a higher median MBF than the cash dividend (share repurchaser) non payer. If true, dummy = 1 otherwise it's zero. A further requirement for a year specific non-zero dummy variable is a minimum of five observations for both payers and non-payers.
Earning Reporting Frequency (ERF)	The frequency (1 to 4 times) at which earnings are reported per annum. 4 = Annual and 1 = Quarterly Reporting.
Cash-Flow Uncertainty (VOL24)	The standard deviation of stock returns over the most recent two year period.
Operating Profitability Volatility (INCV)	The standard deviation of the operating rate of return (*i.e.*, operating income as a percentage of total assets) during the most recent three year period, including the current fiscal year.
Income Risk (SDS)	The standard deviation of the net income during the most recent five year period divided by the most recent year-specific total sales.
Year (YEAR)	Year of Observation.
Constant (CONST)	The intercept of the regression equation.

value zero. The focus, with regard to catering theory, is whether there is a payout (dividend or share repurchase) premium effect and, if so, how this effect varies over time.

Turning now to the life-cycle theory of corporate payout policy, we adopt 4 proxy explanatory variables. First, following, DeAngelo *et al.* [11] we include a proxy explanatory variable for the phase of the life cycle of the firm, the ratio of retained earnings to total equity (RETE) and, in the vein of, Fama and French [15] as well as Grullon and Michaelly [17] we adopt the market value of the firm to reflect firm size (SIZE), another complementary indication of the phase of the life cycle of the firm. The greater the maturity of the individual firm

whether reflected in retained earnings to total equity (RETE) or firm size (SIZE), the greater its expected payout. Fourth, in respect to the development of the firm's set of investment opportunities, we include in our specifications the change in total assets (DAA) following Fama and French [15] and Denis and Osobov [12]. Finally, also following [15] and [12], we adopt the market to book ratio (MBF) to reflect the set of the firm's investment opportunities. The larger the investment opportunity set, the smaller the expected payout.

To assess the empirical importance of the signaling theory of corporate payout policy we turn to our set of 5 proxy explanatory variables. We initially follow Wood [34] and von Eije and Megginson [12] and specify an Earnings Reporting Frequency (ERF) variable, corresponding to the frequency at which earnings are reported, by a firm, per annum. The greater the frequency, the smaller the expected payout and the lower the incentive to payout. Following Lintner [24], Miller and Rock [26] and von Eije and Megginson [14], we also specify an explanatory variable corresponding to the Earnings Ratio (EA). It is computed as the earnings before interest but after tax divided by the book value of total assets. The greater the earnings ratio, the greater the expected payout. Another variable examined, primarily in respect to the signaling theory, is income uncertainty. Anticipated income uncertainty is expected to negatively impact cash dividend payouts due to the expected information content of a subsequent cash dividend decline deteriorating firm value as well as the tendency for external financing to be relatively costly. This latter proxy explanatory variable is operationalised in three ways: (1) income risk (SDS) is computed following von Eije and Megginson [14] as the standard deviation of income during the last 5-years scaled by total sales, (2) operating profitability volatility (INCV) is computed following Chay and Suh [8] as the three-year standard deviation of the operating rate of return and (3) cash-flow uncertainty (VOL24) is computed following Chay and Suh [8] and Lintner [24] as the standard deviation of stock returns during the most recent 3-year period. The greater the income uncertainty, the smaller the expected payout.

In addition we adopt several further control variables. Following von Eije and Megginson [14] we include a lagged return (DPP). There is expected to be a negative relation between this explanatory variable and subsequent payout. Following Fama and French [15] as well as Denis and Osobov [12], we also include in our specifications a year variable (Year), with a view to assessing secular trends over time.

7.3 Methodology

7.3.1 Genetic Programming

Genetic Programming (GP) [22, 28, 30, 31] is an automatic programming technique that employs an Evolutionary Algorithm (EA) to search the space of candidate solutions, traditionally represented using expression-tree structures, for the one that optimises some sort of program-performance criterion. The highly expressive representation capabilities of programming languages allows GP to evolve arithmetic expressions that can take the form of regression models. This class of GP application has been termed "Symbolic Regression", and is potentially concerned with the discovery of both the functional form and the optimal coefficients of a regression model. In contrast to other statistical methods for data-driven modelling, GP-based symbolic regression does not presuppose a functional form, i.e. polynomial, exponential, logarithmic, etc., thus the resulting model can be an arbitrary arithmetic expression of regressors [21]. GP-based regression has been successfully applied to a wide range of financial modelling tasks [7].

GP adopts an Evolutionary Algorithm (EA), which is a class of stochastic search algorithms inspired by principles of natural genetics and survival of the fittest. The general recipe for solving a problem with an EA is as follows. Chose a representation space in which candidate solutions can be specified; design the fitness criteria for evaluating the quality of the solution, a parent selection and replacement policy, and a variation mechanism for generating offspring from a parent or a set of parents. The rest of this section details each of these processes in the case of GP.

In GP, programs are usually expressed using hierarchical representations taking the form of syntax-trees, as shown in Figure 7.1. It is common to evolve programs into a constrained, and often problem-specific user-defined language. The variables and constants in the program are leaves in the tree (collectively named as terminal set), whilst arithmetic operators are internal nodes (collectively named as function set). In the simplest case of symbolic regression, the function set consists of basic arithmetic operators, while the terminal set consists of random numerical constants and a set of explanatory variables termed regressors. Figure 7.1 illustrates an example expression-tree representing the arithmetic expression $x + (2 - y)$.

GP finds out how well a program works by executing it, and then testing its behaviour against a number of test cases; a process reminiscent of the process of black-box testing in a conventional software engineering practice. In the case of symbolic regression, the test cases consist of a set of input-output pairs, where a number of input variables represent the regressors and the output variable represents the regressand. Under this incarnation of program evaluation, GP becomes an error-driven model optimisation procedure, assigning program fitness that is based on some sort of error between the program output value and the actual regressand value; mean squared error being the most prominent form of error employed. Those programs that do well (i.e. high fitness individuals) are chosen to be varied and produce new programs for the new generation. The primary variation operators to perform transitions within the space of computer programs are crossover and mutation.

The most commonly used form of crossover is subtree crossover, depicted in Figure 7.1. Given two parents, subtree crossover randomly (and independently) selects a cross-over point (a node) in each parent tree. Then, it creates two offspring programs by replacing the subtree rooted at the crossover point in a copy of the first parent with a copy of the subtree rooted at the crossover point in the second parent, and vice-versa. Copies are used to avoid disrupting the original individuals. Crossover points are not typically selected with uniform probability. Function sets usually lead to expression-trees with an average branching factor of at least two, so the majority of the nodes in an expression-tree are leaf-nodes. Consequently, the uniform selection of crossover points leads to crossover operations frequently exchanging only very small amounts of genetic material (i.e., small subtrees); many crossovers may in fact reduce to simply swapping two leaves. To counteract this tendency, inner-nodes are randomly selected 90% of the time, while leaf-nodes 10% of the time.

The most commonly used form of mutation in GP is subtree mutation, which randomly selects a mutation point in a tree and substitutes the subtree rooted there with a randomly generated subtree. An example application of the mutation operator is depicted in Figure 7.1. Another common form of mutation is point mutation, which is roughly equivalent to the bit-flip mutation used in genetic algorithms. In point mutation, a random node is selected and the primitive stored there is replaced with a different random primitive of the same rarity taken from the primitive set. When subtree mutation is applied, this involves the modification of exactly one subtree. Point mutation, on the other hand, is typically applied on a per-node basis. That is, each node is considered in turn and, with a certain probability, it is altered as explained above. This allows multiple nodes to be mutated independently in one application of point mutation.

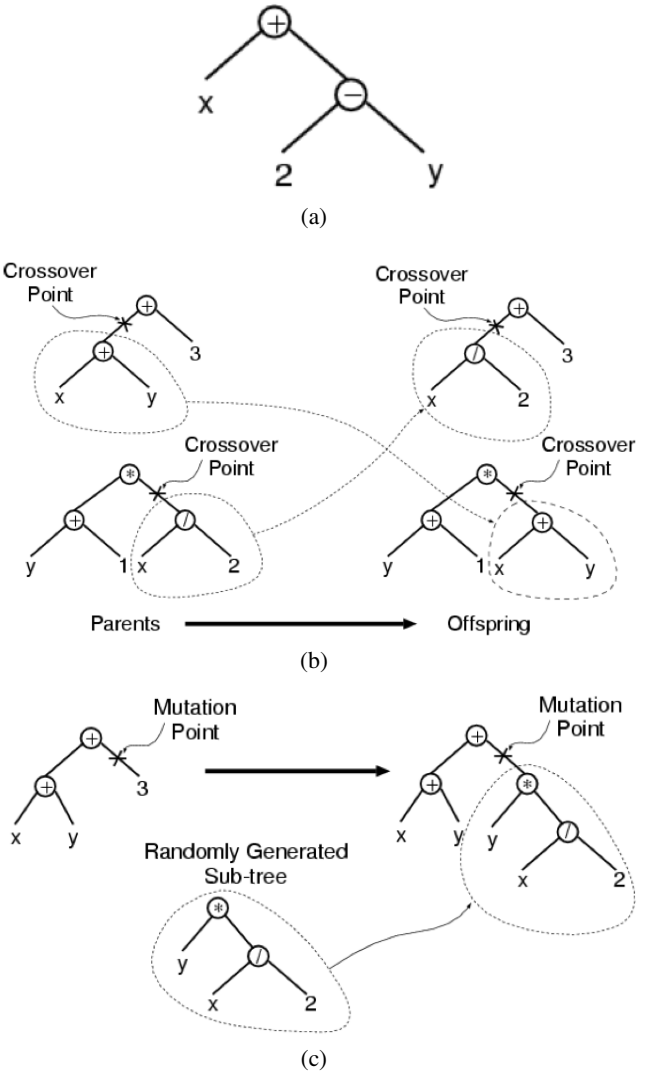


Fig. 7.1 Genetic programming representation and variation operators

Like in any evolutionary algorithm, the initial population of GP individuals is randomly generated. Two dominant methods are the *full* and *grow* methods, usually combined to form the *ramped half-and-half* expression-tree initialisation method [21]. In both the *full* and *grow* methods, the initial individuals are generated so that they do not exceed a user-specified maximum depth. The depth of a node is the number of edges that need to be traversed to reach the node starting from the tree's root node (the depth of the tree is the depth of its deepest leaf). The *full* method generates full tree-structures where all the leaves are at the same depth, whereas the *grow* method allows for the creation of trees of more varied sizes and shapes.

7.3.2 Evolving Symbolic Regression Programs

Our GP algorithm is a standard elitist (the best is always preserved), generational (populations are arranged in generations, not steady-state), panmictic (no program mating restrictions) genetic algorithm with an expression-tree representation. The algorithm uses tournament selection with a tournament size of 7. Root mean squared error (RMSE) is employed as a fitness function. Evolution proceeds for 50 generations, and the population size is set to 1,000 individuals. Ramped-half-and-half tree creation with a maximum depth of 5 is used to perform a random sampling of rules during run initialisation. Throughout the evolution, expression-trees are allowed to grow up to a depth of 8. The evolutionary search employs a combination of crossover, subtree mutation and point mutation; a probability governing the application of each set to 0.5, 0.25 and 0.25 for each operator respectively. We employed a standard single-typed program representation; the function set is consisted of the four basic arithmetic operators (protected division), whereas the terminal set contains the regressors.

7.3.3 Model Overfitting Avoidance

In order to avoid model overfitting [1, 2], we employed a technique that combines the three datasets (training, validation, testing) machine learning methodology, and the objective of minimising the structural complexity of the model [16]. The two sets methodology (training and test dataset) for learning a model using an iterative model-training technique does not prevent by itself overfitting the training set. A common approach is to add a third set – a validation set – which helps the learning algorithm to measure its generalisation ability. Another common practice that has been shown to prevent model overfitting in learning algorithms that employ symbolic model representations is to minimise the model structural complexity. The learning technique uses a validation set to select best-of-generation individuals that generalise well; individuals considered for candidate elitists are those that reside in the Pareto-frontier of a two-objective population ranking of program-size versus the root mean square error (RMSE) criterion.

The initial dataset is randomly segmented into two non-overlapping subsets for training and testing with proportions of 60% and 40% respectively. The training set is further randomly divided into two non-overlapping subsets: the fitness evaluation data-set, with 67% of the training data, and the validation data-set with the remaining 33%. The fitness measure consists of minimising the RMSE on the fitness evaluation data-set. At each generation, a two-objective sort is conducted in order to extract a set of non-dominated individuals (Pareto front) with regards to the lowest fitness evaluation data-set RMSE, and the smallest model complexity in terms of expression-tree size, as measured by the number of tree-nodes. The rationale behind this is to create a selection pressure towards simpler prediction models that have the potential to generalise better. These non-dominated individuals are then evaluated on the validation data-set, with the best-of-generation model designated as the one with the smallest validation RMSE. The use of a Pareto-frontier of canidate elitists reduces the number of individuals tested against the validation set in order to avoid selecting best-of-generation programs that are coincidentally performing well on the validation set. Additionally, such a method allows the learning algorithm to evaluate the generalisation ability of a wide range of accuracy/complexity tradeoffs.

During tournament selection based on the fitness evaluation data-set performance, we used the model complexity as a second point of comparison in cases of identical error rates, thus imposing a bias towards smaller programs though the use of lexicographic parsimony

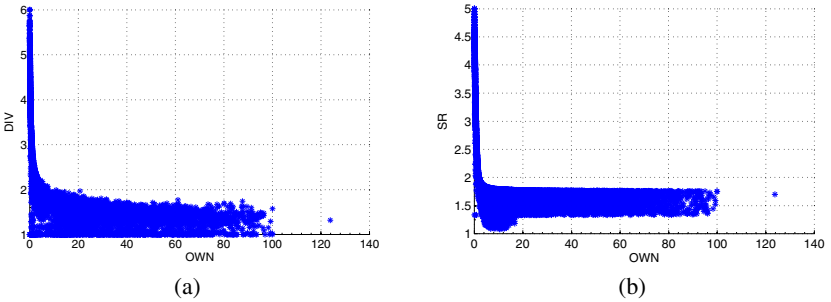
pressure. In every independent evolutionary run, initial dataset segmentation is randomly performed.

7.3.4 Experimental Context

We employ an evolutionary machine learning method to induce models that best describe a regressand variable given a set of input regressor variables. Regressands are the Cash Dividends (DIV) and Share Repurchases (SR), whereas regressors are related to four different theories of corporate payout policy determination, namely, *Agency theory*, *Life-cycle theory*, and *Information-asymmetry signalling theory* as well as *catering* theory, for which a dummy proxy variable is included. Our first aim is to discover accurate symbolic regression models, given different sets of regressor variables defined in each theory separately. Secondly, we are interested in determining whether the evolutionary method will be able to learn models that are in accordance to the conventional theories, by allowing the search algorithm to work on a regressor-space that incorporates all of the regressor variables that are defined in the three conventional theories. This experiment has been specially designed to take advantage from the inherent capability of the GP algorithm to perform feature selection, by allowing the error-minimising search to concentrate of those areas of the model space that contain individuals consisting of the most well-explanatory regressor variables. We performed 50 independent evolutionary runs for each different regressor-setup in order to account for the stochastic nature of the adaptive search algorithm.

7.4 Results

Table 7.2 illustrates the best evolved symbolic regression models using regressor variables from different theories of corporate payout. The first column indicates the regressand variable (either cash dividends, DIV or share repurchases, SR), whereas the second column indicates the regressors which we adopt in relation to each theory of corporate payout. Resulting models have been clustered according to the set of regressor variables defined in each different theory of corporate payout. These optimal solution models indicate a considerable complexity with regard to model specification relative to the variety of classical statistical models adopted in the mainstream literature, particularly the panel regression modelling methodology.



(a) (b)

Fig. 7.2 (a) Model output for DIV vs. OWN; (b) Model output for SR vs. OWN

Figure 7.5 presents a box-plot illustrating the distribution of best-of-run RMSE, indicating the best-fit models from 50 independent evolutionary runs under each different regressor-setup. Contrasting among the corporate payout theories, results suggest that both DIV and SR modelling are more accurately performed using regressor variables defined in Life-cycle theory; the best models attaining a root mean square error (RMSE) of 1.72 and 2.20 respectively. In addition, it is worthwhile observing that, in every instance, the theories of corporate payout substantially outperform in their explanations of cash dividend payouts relative to share repurchases.

In order to quantify the relationship between the use of each regressor variable and the model-output (for the optimal models presented in Table 7.2), we calculated their Pearson correlation coefficient (PCC) via monitoring the model-output in a series of model invocations with model-inputs represented by particular realisations of regressor variables. To complement our investigation of the regressor vs model-output relationship we present the scatter plots of the values that were used to calculate these correlations, for the strongest relationships discovered.

In addition, we present statistics from the average proportion of regressors contained (APU) in the expression-trees of best-of-run individuals, which gives an indication of the relative importance of particular regressors given that the survival and propagation of their embodying model thoughout the evolutionary run signals the evolutionary viability and thus

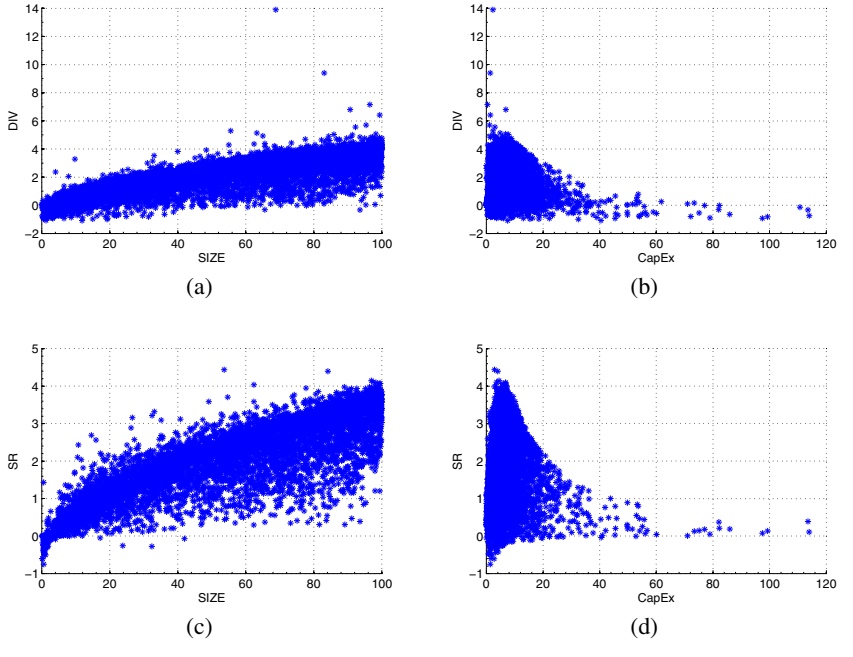


Fig. 7.3 (a) Model output for DIV vs. SIZE; (b) Model output for DIV vs. CapEx; (c) Model output for SR vs. SIZE; (d) Model output for SR vs. CapEx

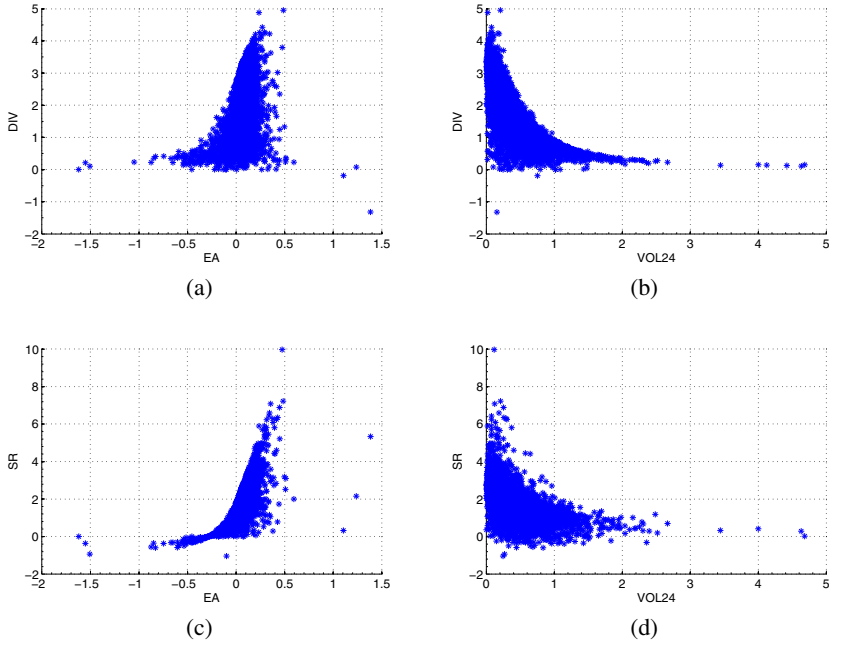


Fig. 7.4 (a) Model output for DIV vs. EA; (b) Model output for DIV vs. VOL24; (c) Model output for SR vs. EA; (d) Model output for SR vs. VOL24

merit of that model. Regressor containment is quantified via the proportion of a particular regressor variable in an expression-tree relative to the rest of the regressors included in that expression-tree.

Table 7.3 collectively presents the Pearson correlation coefficient (PCC) between the regressors of a particular theory and the model-output representing cash dividends (DIV) or share repurchases (SR). In the case of Agency theory, the evolved expressions modelling both cash dividends (DIV) and share repurchases (SR) encapsulate a medium negative correlation between model-output and the regressor concerning the level of concentration of firm ownership (OWN). The level of concentration of ownership (OWN) is a relatively important regressor as evidenced by its pronounced average containment in best-of-run expression-trees. Figure 7.2 presents the corresponding scatter-plots of the values used to calculate the Pearson correlation coefficient (PCC). In the case of Life-cycle theory, the relative value of the firm (SIZE) has a strong positive correlation in both cash dividends (DIV) and share repurchases (SR) modelling. In both cases this particular regressor has a relatively dominant proportion of 18% and 22% in the best-of-run expression trees for DIV and SR models respectively. A weak negative correlation is also found between capital expenditure (CapEx) and model output for both cash dividends (DIV) and share repurchases (SR). Figure 7.3 presents the corresponding scatter-plots of the values used to calculate the Pearson correlation coefficient (PCC).

Finally, in the case of Information-asymmetry signalling theory, strong positive correlations were found between model output and regressor EA, while strong negative correlations were found between model output and regressor VOL24. This finding is consistent in both DIV and SR modelling. Figure 7.4 presents the corresponding scatter-plots of the values used to calculate the Pearson correlation coefficent (PCC).

Table 7.2 Best evolved symbolic regression models

Regressand	Regressors	Model
Agency Theory		
Cash Dividents	OWN, CASH, LR	$\frac{\left(LR+\frac{2*LR}{LR+OWN+(OWN*CASH)}\right)+(OWN+4*LR)}{\frac{(OWN+LR)*2*LR}{\frac{LR}{OWN}+OWN+LR}+OWN+LR}$
Share Repurchases	OWN, CASH, LR	$\frac{2*OWN+3*CASH}{OWN+CASH+\frac{2*(CASH+OWN)}{\frac{CASH}{OWN}+2*OWN}}$
Life Cycle Theory		
Cash Dividents	RnD, RETE, DAA, MBF, SIZE, CapEx, DPP	$\frac{SIZE-CapEx-RnD-DAA-CapEx*DPP}{2*CapEx*DPP^2+RnD+DAA+CapEx+\frac{SIZE+RnD+DAA+SIZE}{RnD+CapEx}}$
Share Repurchases	RnD, RETE, DAA, MBF, SIZE, CapEx, DPP	$\frac{DPP-SIZE-4*CapEx+(DPP-SIZE)*(DAA+DPP)+SIZE*CapEx}{\frac{CapEx+SIZE}{2*MBF}+2*CapEx^2+SIZE}$
Information-Asymmetry Signaling Theory		
Cash Dividents	EA, ERF, VOL24, INCV, SDS	$\frac{EA+ERF+VOL24+(VOL24*SDS)+\frac{2*ERF-VOL24}{(ERF-EA)*(ERF+VOL24)}}{ERF-EA+2*VOL24^2+\frac{SDS}{ERF}*(INCV+ERF)}$
Share Repurchases	EA, ERF, VOL24, INCV, SDS	$\frac{EA*ERF+6*EA+2*ERF}{EA^2-2*EA+2*SDS+VOL24+(ERF+SDS)*ERF+2*SDS^2*INCV}$
Regressors from all theories		
Cash Dividents	OWN, CASH, LR CCD, RnD, RETE, DAA, MBF, SIZE, CapEx, DPP, EA, ERF, VOL24, INCV SDS	$\frac{SIZE-(CASH-VOL24)*(VOL24-ERF)+\frac{RnD+CASH}{OWN+VOL24}}{INCV+CapEx+CASH+\frac{SIZE+LR}{CapEx+CASH}}$
Share Repurchases	OWN, CASH, LR CCD, RnD, RETE, DAA, MBF, SIZE, CapEx, DPP, EA, ERF, VOL24, INCV SDS	$\frac{3*SIZE-CapEx-LR-SIZE*DAA+EA+\frac{SIZE*CCD}{OWN+VOL24}}{\frac{\frac{VOL24}{EA^2}}{\frac{SIZE*VOL24}{CapEX+INCV}}+VOL24*CASH*SDS*ERF+CapEx+SIZE-EA+ERF}$

Very interesting results were also obtained in the second series of experiments which collectively used all proxy explanatory regressor variables defined in the three potentially mutually inclusive theories of corporate payout policy determination. The box-plot depicted in Figure 7.5 suggests that models based on all regressors outperform those based on individual theories; best models attaining a RMSE of 1.65 and 2.12 for DIV and SR modelling respectively. Once again it is evident that these theories exhibit superior explanatory power with respect to cash dividend payouts rather than share repurchases. The most substantial finding was that in order to induce models that are more accurate than the ones based on subsets of regressors, the evolutionary algorithm synthesised expression-trees, which used regressor variables that showed the strongest correlations in previous modelling based on distinct

theories. Throughout an implicit feature selection process, GP was able to identify the most information-rich regressors, which have been proven efficient in different theories of corporate payout, and combine them in an useful, novel way. Once again, in this all-encompassing extended specification, the level of concentration of ownership (OWN), the relative market value of the firm (SIZE), the scaled adjusted earnings variable (EA) and the cash flow uncertainty variable (VOL24) exhibit pronounced effects consistent with the indicated theories of corporate payout. In particular, these regressor variables exhibit the following effects: the level of concentration of firm ownership, OWN (medium negative correlation), the relative firm value, SIZE (strong positive correlation), and the scaled adjusted earnings, EA (medium positive correlation), VOL24 (medium negative correlation). Detailed statistics are depicted is Table 7.3.

Table 7.3 Pearson correlation coefficients (PCC) between the values of regressors used in the best evolved model and the output value of the model. Table also presents the average proportion of regressors containment (APU) in best-of-run individuals. Averages from 50 independent evolutionary runs (std. deviation in parentheses)

	Agency		Life Cycle		Info. Asym. Signal.		All	
Regressor	**PCC**	**APU**	**PCC**	**APU**	**PCC**	**APU**	**PCC**	**APU**
Cash Dividends								
OWN	-0.54	0.33 (0.05)	-	-	-	-	-0.45	0.12 (0.06)
CASH	-0.14	0.23 (0.16)	-	-	-	-	-0.17	0.09 (0.04)
LR	0.13	0.43 (0.15)	-	-	-	-	-0.06	0.12 (0.06)
CCD	-	-	-	-	-	-	0.001	0.10 (0.05)
RnD	-	-	-0.03	0.14 (0.08)	-	-	0.02	0.06 (0.02)
RETE	-	-	0.07	0.05 (0.01)	-	-	n/a	0.0 (0.0)
DAA	-	-	-0.15	0.15 (0.07)	-	-	0.01	0.08 (0.03)
MBF	-	-	0.03	0.09 (0.04)	-	-	0.02	0.11 (0.05)
SIZE	-	-	0.78	0.18 (0.07)	-	-	0.79	0.14 (0.07)
CapEx	-	-	-0.16	0.37 (0.10)	-	-	-0.08	0.10 (0.05)
DPP	-	-	0.03	0.17 (0.07)	-	-	0.13	0.06 (0.01)
EA	-	-	-	-	0.34	0.12 (0.07)	0.20	0.12 (0.08)
ERF	-	-	-	-	-0.04	0.13 (0.03)	0.03	0.12 (0.06)
VOL24	-	-	-	-	-0.75	0.17 (0.07)	-0.39	0.12 (0.07)
INCV	-	-	-	-	-0.12	0.15 (0.12)	-0.17	0.11 (0.05)
SDS	-	-	-	-	-0.10	0.14 (0.06)	-0.07	0.09 (0.06)
Shared Repurchases								
OWN	-0.43	0.43 (0.05)	-	-	-	-	-0.40	0.11 (0.07)
CASH	-0.01	0.40 (0.09)	-	-	-	-	0.02	0.07 (0.02)
LR	-0.03	0.15 (0.07)	-	-	-	-	-0.19	0.11 (0.05)
CCD	-	-	-	-	-	-	0.03	0.09 (0.04)
RnD	-	-	-0.01	0.12 (0.07)	-	-	0.04	0.04 (0.006)
RETE	-	-	0.01	0.13 (0.03)	-	-	n/a	0.0 (0.0)
DAA	-	-	-0.04	0.17 (0.09)	-	-	-0.03	0.07 (0.02)
MBF	-	-	0.04	0.20 (0.10)	-	-	0.03	0.07 (0.01)
SIZE	-	-	0.82	0.22 (0.07)	-	-	0.77	0.17 (0.09)
CapEx	-	-	-0.10	0.36 (0.15)	-	-	-0.08	0.08 (0.02)
DPP	-	-	0.05	0.11 (0.07)	-	-	0.14	0.07 (0.02)
EA	-	-	-	-	0.67	0.37 (0.10)	0.32	0.15 (0.07)
ERF	-	-	-	-	-0.06	0.17 (0.07)	-0.004	0.12 (0.06)
VOL24	-	-	-	-	-0.58	0.15 (0.10)	-0.29	0.09 (0.04)
INCV	-	-	-	-	-0.08	0.15 (0.10)	-0.10	0.12 (0.06)
SDS	-	-	-	-	-0.09	0.19 (0.08)	-0.04	0.07 (0.04)

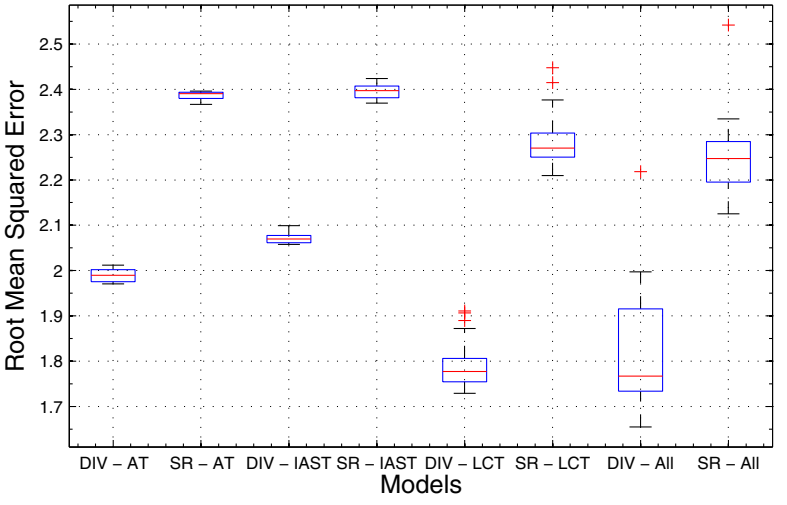


Fig. 7.5 Distribution of Root Mean Squared Errors from best-of-run individuals for different experimental setups. Total cash dividends and Share repurchases have been abbreviated to DIV and SR respectively. Agency Theory has been abbreviated to AT. Life Cycle Theory has been abbreviated to LCT. Information-Asymmetry Signalling Theory has been abbreviated to IAST. "ALL" represents the experiments that took into account all regressors

7.5 Summary and Concluding Remarks

In this Chapter, we have examined the determination of corporate payout policy in the United States during the period 1990 to 2008, using a novel genetic programming methodology to inform our model specifications. Furthermore, our best evolved symbolic modelling specification is unprecedented with regard to the broad range of pertinent proxy explanatory variables included simultaneously in our ultimate, as well as our candidate, model specifications.

Taken together, in line with the mainstream literature, our findings corroboratively provide a preponderance of support for the agency cost-based life-cycle theory of corporate payout policy determination. In addition, proxy variables relating to this latter hybrid hypothesis, with regard to corporate payout policy determination, show strong, theoretically consistent, effects on corporate payout. Specifically, across our modelling specifications, we document a pronounced positive relation between both the relative market value of the firm and the adjusted earnings to assets ratio and measures of corporate payout, cash dividend payouts and share repurchases. In the same vein, we document a compelling negative relation between the level of concentration of firm ownership and cash dividend and share repurchase corporate payouts. In addition, we document a somewhat weaker relation between cash flow uncertainty and these forms of corporate payout. These findings are, on the whole, corroborative with respect to the extant literature. They are important in light of the relatively flexible, genetic programming style, model specification inference methodology, adopted in this Chapter.

Future work in this important area might extend the adopted genetic programming methodology to include constants in the grammar. In addition, it may be helpful to conduct a random effects panel regression model utilising these data to allow a direct comparison of findings

across these methodologies as well as including a contrast of the underlying theoretical frameworks from which these methodologies are derived. We opt to leave this avenue of research for future work.

Acknowledgements. Alexandros Agapitos would like to acknowledge the support of his research activities which is provided by Science Foundation Ireland (Grant number 08/SRC/FM1389).

References

1. Agapitos, A., O'Neill, M., Brabazon, A.: Evolutionary learning of technical trading rules without data-mining bias. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 294–303. Springer, Heidelberg (2010)
2. Agapitos, A., O'Neill, M., Brabazon, A.: Maximum-margin decision surfaces for increased generalisation in evolutionary decision-tree learning. In: Foster, J., Silva, S. (eds.) Proceedings of 14th European Conference on Genetic Programming. LNCS, Springer, Torino (2011)
3. Baker, M., Wurgler, J.: Appearing and disappearing dividends: the link to catering incentives. Journal of Financial Economics 73(2), 271–288 (2004a)
4. Baker, M., Wurgler, J.: A catering theory of dividends. Journal of Finance 59(3), 1125–1165 (2004b)
5. Bhattacharya, S.: Imperfect information, dividend policy and ëthe bird in the hand fallacy. Bell Journal of Economics 10(1), 259–270 (1979)
6. Black, F.: The dividend puzzle. Journal of Portfolio Management 2(3), 5–8 (1976)
7. Brabazon, A., O'Neill, M.: Biologically Inspired Algorithms for Financial Modelling. Natural Computing Series. Springer, Heidelberg (2006)
8. Chay, J.B., Suh, J.: Payout policy and cash-flow uncertainty. Journal of Financial Economics 93(1), 88–107 (2009)
9. DeAngelo, H., DeAngelo, A.: The irrelevance of the mm dividend irrelevance theorem. Journal of Financial Economics 79(2), 293–315 (2006)
10. DeAngelo, H., DeAngelo, A., Skinner, D.J.: Are dividends disappearing? dividend concentration and the consolidation of earnings. Journal of Financial Economics 72(3), 425–456 (2004)
11. DeAngelo, H., DeAngelo, A., Stulz, R.M.: Dividend policy and the earned/contributed capital mix: A test of the lifecycle theory. Journal of Financial Economics 81(2), 227–254 (2006)
12. Denis, D.J., Osobov, I.: Why do firms pay dividends? international evidence on the determinants of dividend policy. Journal of Financial Economics 89(1), 62–82 (2008)
13. Dittmar, A., Mahrt-Smith, J.: Corporate governance and value of cash holdings. Journal of Financial Economics 83(3), 599–634 (2007)
14. Eije, H.V., Megginson, W.L.: Dividends and share repurchases in the european union. Journal of Financial Economics 89(2), 347–374 (2008)
15. Fama, E.F., French, K.R.: Disappearing dividends: Changing characteristics or lower propensity to pay? Journal of Financial Economics 60(1), 3–43 (2001)
16. Gagné, C., Schoenauer, M., Parizeau, M., Tomassini, M.: Genetic programming, validation sets, and parsimony pressure. In: Collet, P., Tomassini, M., Ebner, M., Gustafson, S., Ekárt, A. (eds.) EuroGP 2006. LNCS, vol. 3905, pp. 109–120. Springer, Heidelberg (2006)

17. Grullon, G., Michaely, R.: Dividends, share repurchases, and the substitution hypothesis. Journal of Finance 57(4), 1649–1684 (2002)
18. Grullon, G., Michaely, R., Benartzi, S., Thaller, R.H.: Dividend changes do not signal changes in future profitability. Journal of Business 78(5), 1659–1682 (2005)
19. Jensen, M.C.: Agency costs of free cash flow, corporate finance, and takeovers. American Economic Review 76(2), 323–329 (1986)
20. John, K., Williams, J.: Dividends, dilution, and taxes: A signalling equilibrium. Journal of Finance 40(4), 1053–1070 (1985)
21. Koza, J.: Genetic Programming: on the programming of computers by means of natural selection. MIT Press, Cambridge (1992)
22. Koza, J.R.: Human-competitive results produced by genetic programming. Genetic Programming and Evolvable Machines 11(3/4), 251–284 (2010), doi:10.1007/s10710-010-9112-3; Tenth Anniversary Issue: Progress In Genetic Programming And Evolvable Machines
23. LaPorta, R., de Silanes, F.L., Shleifer, A., Vishny, R.W.: Agency problem and dividend policies around the world. The Journal of Finance 55(1), 1–33 (2000)
24. Lintner, J.: Distribution of incomes of corporations among dividends, retained earnings, and taxes. American Economic Review 46(2), 97–113 (1956)
25. Miller, M.H., Modigliani, F.: Dividend policy, growth, and the valuation of shares. Journal of Business 34(4), 411–433 (1961)
26. Miller, M.H., Rock, K.: Dividend policy under asymmetric information. Journal of Finance 40(4), 1031–1051 (1985)
27. Neely, C., Weller, P., Dittmar, R.: Is technical analysis in the foreign exchange market profitable? a genetic programming approach. Journal of Financial and Quantitative Analysis, 405–426 (1997)
28. O'Neill, M., Vanneschi, L., Gustafson, S., Banzhaf, W.: Open issues in genetic programming. Genetic Programming and Evolvable Machines 11(3/4), 339–363 (2010), doi:10.1007/s10710-010-9113-2; Tenth Anniversary Issue: Progress in Genetic Programming and Evolvable Machines
29. Pinkowitz, L., Stulz, R., Williamson, R.: Does the contribution of corporate cash holdings and dividends to firm value depend on governance? a cross-country analysis. Journal of Finance 61(6), 2725–2751 (2006)
30. Poli, R., Langdon, W.B., McPhee, N.F.: A field guide to genetic programming (With contributions by J. R. Koza) (2008), Published via `http://lulu.com`, freely available at `http://lulu.com`, `http://lulu.com`
31. Poli, R., Vanneschi, L., Langdon, W.B., McPhee, N.F.: Theoretical results in genetic programming: The next ten years? Genetic Programming and Evolvable Machines 11(3/4), 285–320 (2010), doi:10.1007/s10710-010-9110-5; Tenth Anniversary Issue: Progress in Genetic Programming and Evolvable Machines
32. Skinner, D.J.: The evolving relation between earnings, dividends, and stock repurchases. Journal of Financial Economics 87(3), 582–609 (2008)
33. Weston, F., Siu, J.: Changing motives for share repurchases. Finance Paper 3. Anderson Graduate School of Management UCLA, Los Angeles (2003)
34. Wood, A.: Death of the dividend. CFO Europe research report The Economist Group, London (2001)

8

Tackling Overfitting in Evolutionary-Driven Financial Model Induction

Clíodhna Tuite[1,2], Alexandros Agapitos[1,2], Michael O'Neill[1,2], and Anthony Brabazon[1,3]

[1] Natural Computing Research and Applications Group & Financial Mathematics and Computation Research Cluster, Complex and Adaptive Systems Laboratory, University College Dublin, Ireland
[2] School of Computer Science and Informatics, University College Dublin, Ireland
[3] School of Business, University College Dublin, Ireland
cliodhna.tuite@gmail.com, alexandros.agapitos@ucd.ie, m.oneill@ucd.ie, anthony.brabazon@ucd.ie

Summary. This chapter explores the issue of overfitting in grammar-based Genetic Programming. Tools such as Genetic Programming are well suited to problems in finance where we seek to learn or induce a model from data. Models that overfit the data upon which they are trained prevent model generalisation, which is an important goal of learning algorithms.

Early stopping is a technique that is frequently used to counteract overfitting, but this technique often fails to identify the optimal point at which to stop training. In this chapter, we implement four classes of stopping criteria, which attempt to stop training when the generalisation of the evolved model is maximised.

We show promising results using, in particular, one novel class of criteria, which measured the correlation between the training and validation fitness at each generation. These criteria determined whether or not to stop training depending on the measurement of this correlation - they had a high probability of being the best among a suite of potential criteria to be used during a run. This meant that they often found the lowest validation set error for the entire run faster than other criteria.

8.1 Introduction

Overfitting is a commonly studied problem which arises in machine learning techniques such as Genetic Programming (GP). A model is described as overfitting the training data if, despite having a high fit on the training examples, there exists another model which has better fitness on the data as a whole, despite not fitting the training data as well [15].

8.1.1 Causes of Overfitting

There are different reasons why overfitting can occur. For example, the existence of noise in training samples can cause a model to be fit to the data which is more complex than the

A. Brabazon et al. (Eds.): Natural Computing in Comput. Finance, SCI 380, pp. 141–161.
springerlink.com

true underlying model [19]. For symbolic regression, an example would be fitting a high order polynomial to noisy data, which happens to pass through all training points, when the true function is in fact a lower order polynomial. Another cause of overfitting is bias in the training data. Overfitting is more likely to occur when the training sample size is small. The more training data available, the more likely we are to discover the true underlying model, and the less likely we are to settle on a spurious result. Overfitting is also more likely to occur with complex hypotheses [19]. Learning algorithms that are run for a long time are more likely to trigger overfitting, than if they had been run for a shorter time period [3].

Overfitting is a symptom of experimental setups which produce results that fail to generalise beyond the specific environment in which they were trained. While generalisation has traditionally been underexplored in the GP literature, there have been a number of recent papers examining this important issue [3, 6, 9, 11, 19, 22, 25]. Among the techniques proposed to counteract overfitting, popular examples include the use of parsimony constraints, and the use of a validation set. Parsimony contraints are inspired by Occam's Razor and the minimum description length (MDL) principle. The MDL principle claims that the most preferable solution is the one that minimizes the information required to encode it [25]. However, the direct link between parsimony and better generalisation is disputed in [8]. Domingos [8] notes that overfitting is an unwanted side-effect not of complexity itself, but of considering a large number of potential models. This results in a high chance of discovering a model with a high fitness on the training data *purely by chance*. Evaluating a smaller number of more complex models has a lower chance of causing overfitting than evaluating a larger number of simpler models [8].

Early stopping is a popular technique. When using this approach, generalisation is typically measured by observing the fitness of the evolved model on a validation data set, which is separate from the training data set. Training is stopped if the generalisation of the model on this validation set degrades. In this study, we choose to focus on early stopping in order to illustrate its usefulness as a technique to boost generalisation. Furthermore, we propose some extensions to early stopping as it has traditionally applied, via the inclusion of criteria to determine a more intelligent stopping point. These criteria are inspired by work previously carried out in the neural networks literature [21].

8.1.2 Structure of Chapter

The next section provides some background to the issues explored in this chapter, including a brief review of some of the work on generalisation to date. Section 8.3 moves on to work through and explain some examples of overfitting as observed in symbolic regression problems tackled using Grammatical Evolution, a form of grammar-based GP. Section 8.4 first introduces stopping criteria which can be used to enhance the technique of early stopping, which is used to avoid overfitting. Section 8.5 applies these criteria to trading rule problems tackled using an alternative form of grammar-based GP. Finally, Section 8.6 concludes the chapter.

8.2 Background

GP is a stochastic search and optimisation technique. It operates by first generating a "population" of random solutions, composed from elements of a function and terminal set. The terminal set is composed of a list of external inputs (typically variables), and constants that can appear in the solution. It may also contain functions with no arguments - an example

would be a function which returns a different random number with each invocation. The contents of the function set vary between problem domains - an example of such operators in a simple numeric problem are arithmetic operators. Solutions are iteratively refined over a period of some number of pre-specified generations, using the concept of a fitness function to guide search, and the operators of crossover and mutation to transition across the search space. Crossover works by combining elements of two "parent" solutions, in order to further explore the search space. Mutation randomly alters a part of a randomly selected solution among the population of solutions, in order to promote diversity in the population, and thus explore new areas of the search space. For a more detailed introduction to GP, see [2] or [20].

8.2.1 Previous Attempts to Tackle Overfitting in Genetic Programming

Two Data Sets Methodology

In a 2002 paper, [11] highlights the importance of using a separate testing set alongside a training set when evolving solutions to learning problems. The author also advocates the use of formal guidelines when selecting the training and testing cases. Such guidelines might stipulate rules for selecting training and testing instances to ensure the representativeness of the training cases, and to determine the degree of overlap between the two data sets [11]. In a case of simulated learning (the Santa Fe ant trail), it is shown that training on a given trail by no means guarantees the generalisation of the solution to other trails using the same primitives and rules for trail construction. In order to produce results that generalise to a particular class of trails, [11] conducted experiments using 30 training trails and 70 testing trails, with fitness being evaluated by counting the number of pieces of food gathered over each of the 30 training trails.

Validation Sets and Parsimony

In [22], Thomas and Sycara use a GP-based system to discover trading rules for the Dollar/Yen and Dollar/DM markets. They reserved a particular concern for the aspects of the system that allowed them to fight overfitting, given the inherent noise in financial data. They examined the interactions between overfitting, and both rule complexity and validation methodologies. Excess returns were used as a measure of fitness. The success of the evolved model on the test set was calculated after training had completed. The use of the test set was important given that the focus of this work was on generalizaton.

The authors thought that while controlling the size of trees could negatively impact on the representational capability of the generated rules, it could also reduce the potential for overfitting, and wished to test this hypothesis. Results showed that between trees with maximum tree-depths of 2, 3 and 5, trees of maximum-depth 2 produced superior excess returns, particularly in the Dollar/Yen case [22].

The authors used a validation set to determine when to cut off search in all experiments - evolution was stopped when the average rule fitness on the validation set started to decrease. In later experiments they used the validation set in two additional ways. After search had stopped, the best rule on the training set was identified and was kept if it produced positive returns on the validation set. If not, it was discarded and search resumed from scratch. A second approach examined the fitness of *every rule* from the population on the validation set after search had been stopped, keeping those that produced positive returns.

These two additional approaches did not increase the percentage of excess returns on the test set after training stopped, which the authors found surprising. They attributed this to a lack of correlation between the training and test performances. It is interesting that the authors examined the correlation between the training and test performances as an ex-post diagnostic tool. Below, we describe measuring the correlation between training and *validation* fitnesses, but we use it instead as a potential stimulus to prematurely halt our search. The authors concluded that using the validation methods they described was not providing adequate improvements to excess returns, as evaluated on the test set. They felt that additional criteria needed to be investigated in order to identify a cut-off point for search [22].

Gagné and co-authors [9] also investigate the use of three datasets (training data, validation data, and test data). They evolve solutions to binary classification problems using Genetic Programming. The inclusion of a validation set, to periodically check the evolved a model for a loss of generalisation, reduces the amount of data used to train the model. This reduction in the data reserved for training means that the training algorithm has less information with which to fit a model, and increases the possibility that the model produced will not be representative of the true underlying model. The trade-off between the inclusion of validation data on the one hand, and the reduction in training data on the other hand, is examined by the authors.

Simple solutions are sometimes postulated to both reduce the effect of bloat (an uncontrolled increase in the average size of individuals evolved by GP [13]), and produce solutions without overfitting. The direct link between parsimony on the one hand, and solutions that do not overfit the data on the other hand, has been disputed in [8], as noted in Section 8.1.1. Results that contradict this - for example those results described in [22] and summarised above, have meant that researchers have continued to pursue parsimonious solutions, albeit with an awareness of the challenge made against their effectiveness (for example, in [3]).

Gagné and co-authors [9] also investigate the use of lexicographic parsimony pressure to reduce the complexity of the evolved models. Lexicographic parsimony pressure [12] involves minimizing the error rate on the entire training set, and using the size as a second measure to compare when the error rates are exactly the same. The best individual of a run is the individual who exhibits the lowest error rate on the training set, and the smallest individual is selected in the case of a tie. They establish that while there is no clear advantage in terms of test set accuracy from using a validation set and parsimony pressure, the inclusion of a validation set and parsimony pressure does *lower the variance* of the test set error on the evolved model across 100 runs. They express a desire to develop new stopping criteria in future work, which would be based on the difference between training and validation fitness.

8.2.2 Early Stopping

Early stopping is a method used to counteract overfitting, whereby training is stopped when overfitting begins to take place. In order to enable early stopping, the data is split into three subsections: training data, validation data, and testing data. Initially, a model is fitted to the training data. At regular intervals as the model fitting proceeds, the fitness of the model on the validation data (which has not been used to train the model) is examined for disimprovement. The ability of the evolved model to generalise beyond the training examples is therefore measured while the run is in progress, the assumption being that if the learned model is generalising well, it should exhibit high fitness on both the training data, and the unseen validation data. When a disimprovement is observed in the fitness of the evolved model on the validation data (or is observed for a specified number of consecutive generations or iterations) training is stopped. The model with the lowest validation error prior to training having been stopped, is used as the output of the run [10].

A More Robust Way to Determine When to Stop Training

This simple early stopping technique has been criticised by Prechelt in [21], where he examines the use of early stopping when training a neural network. Prechelt states that the validation set error, in most cases, fluctuates throughout the course of the run. He describes how the validation set error rarely monotonically improves during the early stage of the run, before monotonically disimproving after overfitting has begun to take place. He states that real validation error curves almost always have more than one local minimum. The question then becomes, when should early stopping take place? He goes on to propose three classes of stopping criteria, with the aim of developing criteria which lead to both lower generalisation error, and exhibit a reasonable trade-off between training time and improved generalisation.

8.2.3 Grammatical Evolution: A Brief Introduction

Grammatical evolution (GE) [7, 17] is a popular form of grammar-based GP [14]. A particular strength of GE is the use of a grammar to incorporate domain knowledge about the problem we are attempting to solve. In GE, the process of evolution first involves the generation of a population of randomly-generated binary (or integer) strings, the genotype. In the case of binary genomes, each set of B bits (where traditionally B=8) is converted into its equivalent integer representation. These integer strings are then mapped to a phenotype, or high-level program or solution, using a grammar, which encompasses domain knowledge about the nature of the solution. Therefore, a GE genome effectively contains the instructions of how to build a sentence in the language specified by the input grammar. Grammatical Evolution has been applied to a broad range of problems, including many successful examples in financial modelling [5].

The grammar used in the first set of experiments we performed can be found in Fig. 8.1. The grammar is composed of non-terminal and terminal symbols. Terminals (for example, arithmetic operators) appear in the solution, whereas non-terminals can be further expanded into terminals and non-terminals. Here we can see the syntax of the solution (that will be constructed from arithmetic operators, mathematical functions, variables and constants) is encoded in the grammar.

The mapping process involves the use of an integer from the genotype to choose a production rule from the choices available to the non-terminal currently being mapped. This process proceeds as follows. The first integer from the integer representation of the genotype is divided by the number of rules in the start symbol (`<expr>` in our example). The remainder from this division is used to select a rule from the grammar:

$$\begin{aligned} rule = \; & (\text{Codon integer value}) \\ & MOD \\ & (\text{Number of rules for the current non} - \text{terminal}) \end{aligned}$$

Groups of production rules are indexed from zero, so if the result of division leaves a remainder of 0, the production rule with an index value of zero will be chosen. For example, given there are 5 choices of production rule available to map from `<expr>`, if the first integer in the integer-representation of the genotype was 8, then

$$8 \; MOD \; 5 = 3$$

```
<prog> ::= <expr>
<expr> ::= <expr> <op> <expr>            (0)
         | ( <expr> <op> <expr> )        (1)
         | <pre-op> ( <expr> )           (2)
         | <protected-op>                (3)
         | <var>                         (4)
<op> ::= +                               (0)
       | *                               (1)
       | -                               (2)
<protected-op> ::= div( <expr>, <expr>)
<pre-op> ::= sin                         (0)
           | cos                         (1)
           | exp                         (2)
           | inv                         (3)
           | log                         (4)
<var> ::= X                              (0)
        | 1.0                            (1)
```

Fig. 8.1 Grammar used in Symbolic Regressions

and so the third rule (indexing from zero), which is <protected-op>, would be selected. <protected-op> only contains one possible mapping - div(<expr>,<expr>). This means there is no need to read an integer from the genotype to determine which rule will be chosen to map from <protected-op>, since there is no choice to be made. The next integer in the genotype would then need to be read in order to map between the leftmost <expr> and one of its constituent rules. Let's assume this next integer had value 39. Once again, there are five choices available to map from <expr>, so

$$39 \; MOD \; 5 = 4$$

would select the fourth production rule for <expr>, which counting from 0, is <var>. The third integer in the genotype would next be used to map between <var> and one of its constituent rules. This process continues until either all integers in the genotype have been used up, or our mapping process has resulted in the production of a phenotype (that is a structure comprised only of terminal symbols) [17].

8.3 Some Experiments to Illustrate Overfitting in Symbolic Regression Problems

In order to clearly illustrate the problem of overfitting in model induction using GP, we show in this section what happens when three symbolic regression functions are fit using Grammatical Evolution [17] in a range that biases the output towards an overfit model. This exposition is based on work we carried out in [23].

8.3.1 Setup

Equations 1 through 3 show the target functions. The training dataset, validation data set, and test datasets were all comprised of 10 randomly generated points. The test dataset was not used to train the model, and was comprised of points solely outside the training range in order to specifically focus on the extrapolation capabilities of the evolved model.

Target Function 1:

$$Y = 0.6X^3 + 5X^2 - 10X - 25 \tag{8.1}$$

Training dataset range: [-5, 5].
Validation data set range: [-5, 5].
Test dataset ranges: [-10, -5] and [5, 10].

Target Function 2:

$$Y = 0.3X \times \sin 2X \tag{8.2}$$

Training dataset range: [-1, 1].
Validation dataset range: [-1, 1].
Test dataset ranges: [-2, -1] and [1, 2].

Target Function 3:

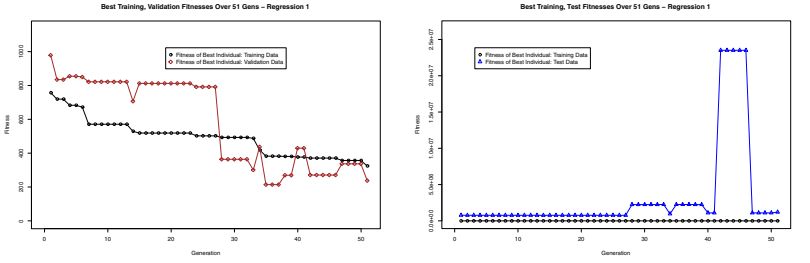
$$Y = \exp X - 2X \tag{8.3}$$

Training dataset range: [-2, 2].
Validation dataset range: [-2, 2].
Test dataset ranges: [-4, -2] and [2, 4].

These functions and ranges were chosen so that the target function would be trained using a biased data sample. The bias resulted from training in a range in which the target function closely resembled an alternative function. Over a wider range than that from which the training data was drawn, the target function looked quite different from this alternative (for example, function 2 is a Sine function as we can see from the target function given above. However, if examined only in the training range of between minus one and one, it resembled a quadratic function, (see Fig. 8.6)). In this way, we engineered a situation in which overfitting was likely to take place. In each case, Grammatical Evolution was run on a population size of 100 individuals, for 51 generations, using Grammatical Evolution in Java [16]. The grammar used is shown in Fig. 8.1.

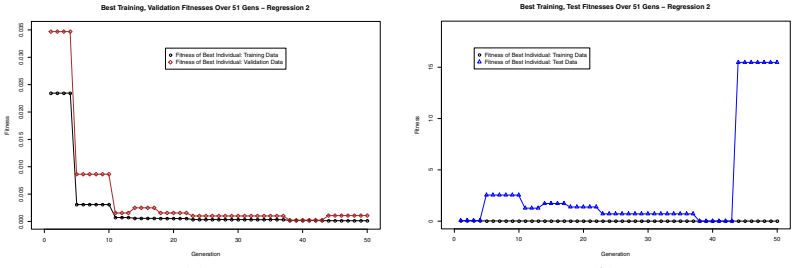
Fitness was evaluated by computing the mean squared error of the training points when evaluated on each individual (therefore the lower the fitness value, the better the evolved function fitted the training data).

$$MSE = \frac{\sum_{i=1}^{n} |targetY - phenotypeY|^2}{n} \tag{8.4}$$



(a) (b)

Fig. 8.2 Target Function 1

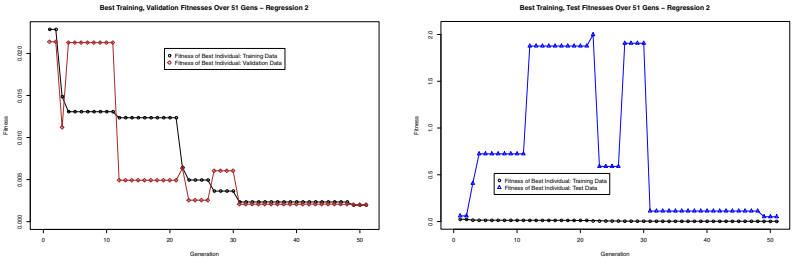


(a) (b)

Fig. 8.3 Target Function 2, Example 1

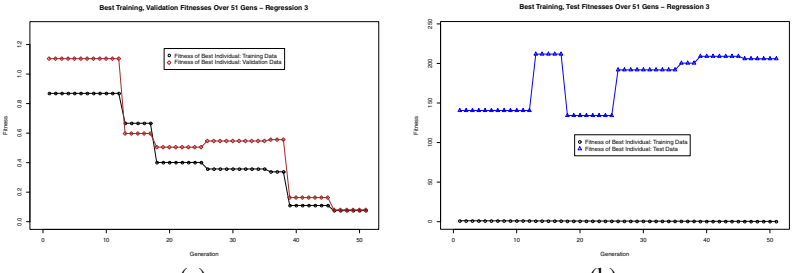
8.3.2 Results

Figs. 8.2 through 8.5 are plots of the fitness of the best individual at each generation as evaluated on the training data, against the fitness of the best individual at each generation as evaluated on the validation and test datasets, for four illustrative runs - one run each of target functions 1 and 3, and two runs of target function 2. Table 8.1 shows results of interest with respect to the fitness as evaluated on the validation and test dataset, for 9 runs. It shows that stopping evolution before the specified number of generations had elapsed, in the majority of cases would have led to the model extrapolating better beyond the range in which it was trained [23].



(a) (b)

Fig. 8.4 Target Function 2, Example 2



(a) (b)

Fig. 8.5 Target Function 3

Table 8.1 Results of Interest: Validation, Test fitnesses

Target Function	Would Traditional Early Stopping Have Been Useful? - Validation Fit.	Would Traditional Early Stopping Have Been Useful? - Test Fitness	Generation of Best Result (Test Fitness) before or after the Gen. of Result of Training, as per Traditional Early Stopping?
1	No	Yes	After
1	No	Yes	Before or Same Time
1	Yes	Yes	After
2	No	Yes	After
2	No	No	After
2	No	Yes	After
3	No	Yes	Before or Same Time
3	No	Yes	After
3	Yes	Yes	Before or Same Time

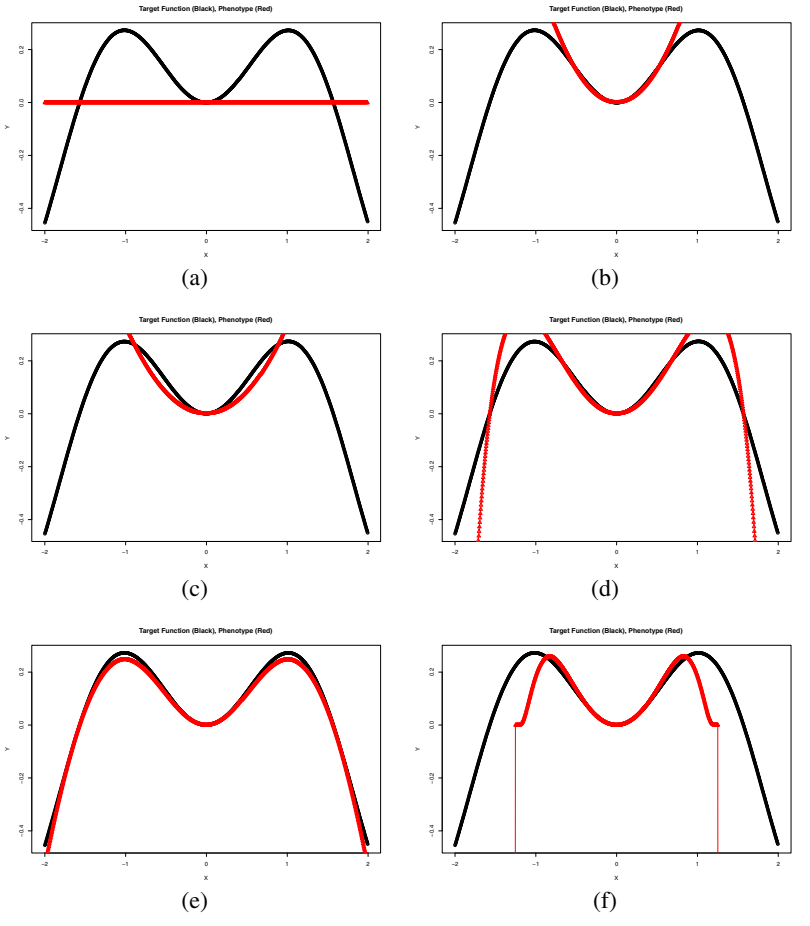


Fig. 8.6 (a) Generation 1 (b) Generation 5 (c) Generation 11 (d) Generation 23 (e) Generation 38 (f) Generation 44

Early stopping has been described in Section 8.2.2. The validation dataset is not used to train the model, but instead is used to test the fitness of the model every once in a while (for example each generation, or at five generation intervals). If the fitness of the best individual as evaluated on the validation dataset disimproves, this is taken as an indication that the evolved model is overfitting the data, and evolution is stopped. Test data is used as before to evaluate the fitness of the evolved model on out-of-sample data, after evolution has terminated, either prematurely (if early stopping has been deemed necessary), or after the specified number of generations has elapsed.

Since we explicitly chose target functions and ranges with an inherent bias, these symbolic regressions triggered overfitting, as expected. Using traditional early stopping, training is stopped the first time validation fitness disimproves. The result of training is taken to be the model produced at the generation immediately before training is stopped. In the second

and third columns of Table 8.1, we look at the validation and test fitnesses (respectively), and determine whether or not performing traditional early stopping would have produced a model with lower validation (column 2) and test set fitnesses (column 3), than the model produced at the end of the run. In eight of the nine runs described, the test fitness was better the generation immediately before the validation fitness first disimproved, than at the end of the run (column 3). Had we stopped evolution the first time the validation fitness disimproved, we would have produced a model that extrapolated better beyond the training range, than that produced at the end of the run (remember that the test set is comprised of points solely outside the training range). We see that in only two of the nine runs was the validation fitness better the generation before it first disimproved than at the end of the run. This is not that surprising. The validation data points are drawn from the same range as the training points. Therefore, overfitting is less likely to occur in the training/validation range than outside of this range.

Prechelt [21] shows that when training artificial neural networks, the first time the error on the validation set increases is not necessarily the best time to stop training, as the error on the validation set may increase and decrease after this first disimprovement. The last column of Table 8.1 shows for each run, if the best stopping point came before or after the generation of the result of training as dictated by traditional early stopping. The best stopping point here refers to the earliest generation of lowest *test* error. We examine the eight runs where the model that we would have evolved using traditional early stopping had better test fitness than the model that would have been evolved at the end of the run. In five out of these eight runs, the optimal generation at which to stop (as measured by test fitness) came later than the generation of the result of training using traditional early stopping [23].

In order to give further insight into the evolutionary process that underlie the changes in fitness observed for the training and test data sets, the phenotype was plotted against the target function in the entire data set range (that is, throughout the range of training, validation and test data), at each generation. Fig. 8.6 shows a selection of these generational graphs for the first run of function 2.

Comparing Figs. 8.6 and 8.3(b), we can clearly see the correspondences between changes in the graphed phenotype over the generations and changes in the fitness as evaluated on the test data. Between generations 1 and 22, the test fitness is either disimproving, or not improving by much. At generation 23 (Fig. 8.6(d)) fitness improves significantly, and at generation 38 (Fig. 8.6(e)), an extremely fit individual has been evolved, both with respect to the training and test set. In Fig. 8.3(b) we see that the error on the test data is extremely low at generation 38. The model extrapolates well. However, come generation 44 (Fig. 8.6(f)), a much less fit function has been evolved. It's fitness on the training data has improved, but it's fitness on the test data has drastically disimproved. In Fig. 8.3(b) we can see an explosion in the test error towards the end of the run [23], which contrasts with the low value of the training error.

8.4 Solutions to Prevent Overfitting - Stopping Criteria

In [21], Prechelt implements three classes of stopping criteria, which determine when to stop training in order to preserve the generalisability of the evolved model. Each class of criteria assumes that we are evaluating the fitness of a model by examining the error of the model on the training and validation datasets. Validation set error is used to measure how well the model is generalising beyond the training examples. *Generalisation loss* is measured by dividing the average validation set error per example measured after the current epoch (E_{va}), by the minimum validation set error observed up until the current epoch (E_{opt}). In our implementation of these criteria, we instead use the validation set error of the best individual

at the current generation, for E_{va}. In percentage terms, the generalisation loss (GL) at epoch t is given by:

$$GL(t) = 100 \times \left(\frac{E_{va}}{E_{opt}} - 1 \right) \tag{8.5}$$

First Class of Stopping Criteria - Stop when the Generalisation Loss Exceeds a Threshold

The first class of stopping criteria measures the loss of generalisation of the trained model at each epoch, and stops training if the generalisation loss is observed to have crossed a predefined threshold (denoted here by α). In [21] the class GL_α is defined as follows:

$$GL_\alpha : \text{stop after first epoch } t \text{ with } GL(t) > \alpha \tag{8.6}$$

While stopping once the generalisation loss passes a certain threshold appears to be a good rule of thumb, it may be useful to add a caveat to that. Before applying early training stopping, we may wish to check whether or not training is still progressing rapidly. The *training progress* is examined over k generations. It measures how much the average training error during the strip of length k, was larger than the minimum training error during the strip. It is given (in per thousand) by:

$$P_k(t) = 1000 \times \left(\frac{\sum_{t'=t-k+1}^{t} E_{tr}(t')}{k \times \min_{t'=t-k+1}^{t} E_{tr}(t')} - 1 \right) \tag{8.7}$$

where $E_{tr}(t)$ is the average training error per example measured after time t. In our experiments using these criteria, $E_{tr}(t)$ measured the training error of the best individual at the current generation.

Second Class of Stopping Criteria - Quotient of Generalisation Loss and Progress

If training is progressing well, then it may be more sensible to wait until both generalisation loss is high and training progress is low, before stopping. The intuition behind this lies in the assumption that while training is still rapidly progressing, generalisation losses have a higher chance to be 'repaired'.

A second class of stopping criteria was defined in [21] to use the quotient of generalisation loss and progress:

$$PQ_\alpha : \text{stop at first end}-\text{of}-\text{strip epoch } t \text{ with } \frac{GL(t)}{P_k(t)} > \alpha \tag{8.8}$$

Third Class of Stopping Criteria - Stop Training when the Generalisation Error Increases in *s* Successive Strips

The third and final class of stopping criteria identified in [21] approached the problem from a different angle than the first two. It recorded the sign of the changes in the generalisation error, and stopped when the generalisation error had increased in a predefined number of successive strips. The magnitude of the changes was not important, only a persistent trend in the direction of of those changes. The third class of stopping criteria is defined as:

$$\begin{aligned} &\mathrm{UP_s}: \text{stop at epoch t iff } \mathrm{UP_{s-1}} \text{ stopped at epoch } \mathrm{t-k} \\ &\text{and } \mathrm{E_{va}} > \mathrm{E_{va}(t-k)} \\ &\mathrm{UP_1} \ : \ \text{stop at first end} - \text{of} - \text{strip epoch t} \\ &\text{with } \mathrm{E_{va}(t)} > \mathrm{E_{va}(t-k)} \end{aligned} \tag{8.9}$$

where s is the number of successive strips.

Stopping criteria decide to stop at some time t during training, and the result of the training is then the set of weights (in the case of neural networks), or the evolved model (in the case of GE) that exhibited the lowest validation error prior to the time at which training was stopped. The criteria can also be used if we are trying to maximise a fitness metric, rather than minimise an error function. In this case the formulae used need to be adjusted to reflect the fact that we are maximising fitness, not minimising error.

8.5 Investigations: Stopping Criteria applied to a Financial Dataset

In Section 8.2 we outlined the contribution of various authors investigating overfitting and generalisation in GP and neural networks. Some of these [9, 21, 22] point to the need for additional criteria to determine the point at which training is stopped to avoid overfitting. Thus inspired, we present an implementation of the stopping criteria detailed in Section 8.4 using an alternative form of Grammar-based GP to GE.

These criteria have not, to the best of our knowledge, been applied in a GP setting before now. This presents a valuable opportunity to apply approaches implemented in the neural networks literature, to the field GP. This work is related to work we carried out using early stopping criteria with symbolic regression in [24]. We also implement an additional stopping criteria not found in the work carried out in [21].

Here, technical trading rules are evolved in the form of decision-trees (DTs) for the trading rule induction problem, in order to generate signals to take a short or long position. We emphasise that our focus is not on the construction of high-quality trading rules, instead we are focussing on demonstrating the practical application and potential usefulness of stopping criteria used to enhance the generalisation of solutions evolved using GP.

8.5.1 Grammar-based GP Experimental Setup

The grammar in Fig. 8.8 presents the grammar adopted for program representation. Each expression-tree is a collection of `if-then-else` rules that are represented as a disjunction of conjunctions of constraints on the values of technical indicators.

Technical Analysis (TA) has been widely applied to analyse financial data and inform trading decisions. It attempts to identify regularities in the time-series of price and volume information, by extracting patterns from noisy data [5]. The technical indicators we used for these experiments are: (a) **simple moving average** (MA), (b) **trade break out** (TBO), (c) **filter** (FIL), (d) **volatility** (VOL), and **momentum** (MOM). For a good introduction to these, please refer to [5]. TA indicators are parameterised with lag periods, which is the number of past time-steps that each operator is looking at. Currently, we allow periods from 5 to 200 closing days, with a step of 5 days. We also include the closing price of the asset at each time-step.

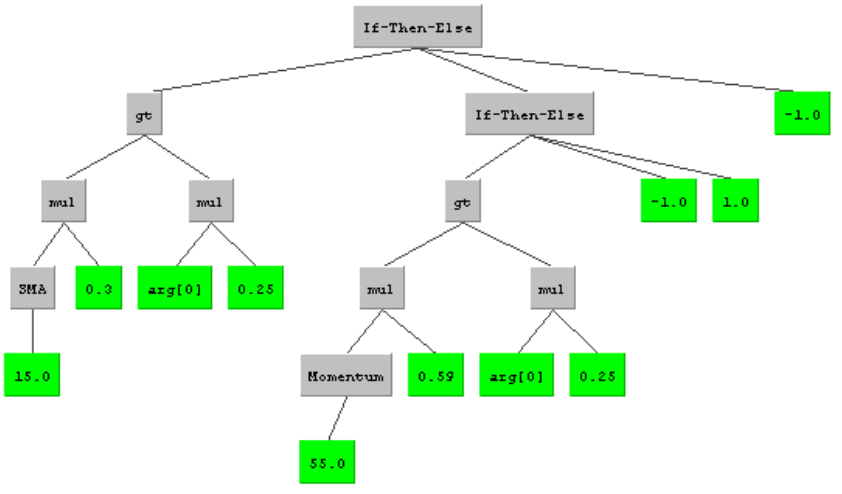


Fig. 8.7 A Sample Decision Tree

A sample decision tree is given in Fig. 8.7. Here, `arg[0]` represents the closing price. This rule tells us to examine if three-tenths of the Simple Moving Average over the past fifteen time-steps is greater than one quarter of the closing price (in essence, a form of momentum indicator). If it is, we next execute the middle branch of the tree and act accordingly, otherwise we take a short position. In order to execute the middle branch of the tree, we evaluate whether or not the momentum over the last 55 time-steps, multiplied by 0.59, is greater than one quarter of the closing price. If it is, we take a short position - otherwise we take a long position.

The GP algorithm employs a panmictic, generational, elitist Genetic Algorithm. The algorithm uses tournament selection. The population size is set to 500, and evolution continues for 50 generations. Ramped-half-and-half tree creation with a maximum depth of 5 is used to perform a random sampling of DTs during run initialisation. Throughout evolution, expression-trees are allowed to grow up to a depth of 10. The evolutionary search employs a variation scheme that combines mutation with standard subtree crossover. A probability governs their application, set to 0.7 in favour of mutation. No cloning was used. The maximisation problem employs a fitness function that takes the form of average daily return (ADR) generated by the rule's trading signals over a training period.

8.5.2 Financial Time-Series and Trading Methodology

The datasets used are daily prices for a number of financial assets. These are the foreign exchange rate of EUR/USD, the Nikkei 255 index, and S&P 500 index, for the period of 01/01/1990 to 31/03/2010. The first 2,500 trading days are used for training, and the remaining 2,729 are equally divided between validation and test sets, both with size of 1,364 days.

Each evolved rule outputs two values, 1 and -1, interpreted as a long and short position respectively. The average return of a rule (Average Daily Return (ADR)) is generated as follows. Let r_t be the daily return of the index at time t, calculated using $(v_t - v_{t-1})/v_{t-1}$, where v_t and v_{t-1} are the values of the time-series at time t and $t-1$ respectively. Also, let

```
<prog> ::= <if>
<if> ::= <predicate> <expr> <expr>
<expr> ::= <if>
         | <signal>
<signal> ::= -1
           | 1
<predicate> ::= <tiexpr> <comp> <arithexpr>
<tiexpr> ::= <arithexpr> <arithop> <tiexpr>
           | <tiexpr> <arithop> <tiexpr>
           | <coeff> * <ti>
           | <coeff> * <ti> + <coeff> * <ti>
           | <ti>
<arithexpr> ::= <val> <arithop> <val>
              | <val>
<valA> ::=  <coeff>
         | closingprice
<coeff> ::= <const> <arithop> <const>
          | <const>
<comp> ::= <
         | >
<arithop> ::= +
            | -
            | *
            | /

<ti> ::= MA ( 5 )
       | ...
       | MA ( 200 )
       | TBO ( 5 )
       | ...
       | TBO ( 200 )
       | FIL ( 5 )
       | ...
       | FIL ( 200 )
       | MOM ( 5 )
       | ...
       | MOM ( 200 )
       | VOL ( 5 )
       | ...
       |VOL ( 200 )
<const> ::= random constant in [-1.0, 1.0]
```

Fig. 8.8 Grammar used on Financial Datasets

s_{t-1} be the trading signal generated by the rule at time $t-1$. Then $d_t = s_{t-1} r_t$ is the realised return at time t. Using a back-test period, an average of d_t can be induced; to annualise this we simply multiply it by 200 trading days. Trading, slippage and interest costs are not considered.

8.5.3 Results and Discussion

The thresholds used for the GL, PQ and UP criteria were closely mirrored on those used in [21], with a few minor changes. The strip length used was also taken from [21], and was set to 5, for both the UP and PQ classes of criteria.

As noted in Section 8.5.1, the evolution of trading rules is governed by a fitness function that maximises the average daily return (ADR) generated over a period. One point to note here is that the formulae have been re-written in terms of the fitness being maximised, rather than the equivalent form of the error being minimised. It is worth pointing out, in order to avoid confusion, that instead of stopping when the generalisation error increased for some number of successive strips, when dealing with fitness maximisation, the 'UP' criteria stopped when the generalisation error *decreased* for some number of successive strips.

Tables 8.2, 8.3 and 8.4 examine the performance of the GL, PQ and UP stopping criteria on the datasets from the EUR/USD, Nikkei 255, and S&P 500 indices. In addition, a new class of criteria were added, which depended on the correlation between the training and validation fitnesses. To the best of our knowledge, these criteria have not been implemented in the same way before. They caused training to be stopped if the Pearson's correlation coefficient between the training set fitness and the validation set fitness, fell below a predefined threshold. The correlation criteria were evaluated at every generation during the run, until a stopping

Table 8.2 Criteria Performance on Euro/USD: Averages and (Standard Deviations) Over 50 Runs

Criterion	Slowness	Generalisability	Prob Best Crit	Prob Halting
GL_1	3.37 (2.7)	1.13 (0.78)	0	0.3
GL_2	3.39 (2.69)	1.13 (0.79)	0	0.28
GL_3	3.41 (2.68)	1.14 (0.78)	0	0.26
GL_5	3.41 (2.68)	1.14 (0.78)	0	0.26
$PQ_{0.5}$	4.08 (4.49)	1.02 (1.76)	0	0.86
$PQ_{0.75}$	4.28 (4.56)	1.03 (1.81)	0	0.86
PQ_1	4.45 (4.61)	1.05 (1.84)	0	0.82
PQ_2	4.8 (4.77)	1.01 (1.77)	0	0.8
PQ_3	5.02 (4.85)	0.94 (0.54)	0	0.8
UP_2	3.29 (4.22)	1.39 (3.6)	0.02	0.56
UP_3	3 (3.56)	1.19 (0.41)	0	0.24
UP_4	3.67 (3.51)	1.34 (0.58)	0	0.06
$COR_{0.8}$	3.62 (4.53)	0.76 (3.62)	0.4	1
$COR_{0.6}$	3.98 (4.66)	0.83 (3.18)	0.18	1
$COR_{0.4}$	4.31 (4.75)	0.88 (2.84)	0.2	0.98
$COR_{0.2}$	4.73 (4.56)	0.97 (2.28)	0.06	0.94
COR_0	5.03 (4.77)	0.93 (1.16)	0.06	0.9
$COR_{-0.2}$	5.73 (5.16)	0.94 (0.94)	0.02	0.76
$COR_{-0.4}$	5.43 (4.62)	0.95 (1.06)	0.02	0.66
$COR_{-0.8}$	5.25 (5.55)	0.58 (2.44)	0.04	0.32

generation was identified. The thresholds used for the correlation criteria were 0.8, 0.6, 0.4, 0.2, 0, −0.2, −0.4 and −0.8.

We experimented with a wide range of correlation values, from values describing strong positive correlation to strong negative correlation. It is important to keep in mind that these values represented thresholds, *which if crossed, would trigger an end to training*. As such, examining whether or not weaker (negative) as well as stronger (positive) thresholds would prove a useful stimulus to stop training merited investigation. As can be seen in Tables 8.2 to 8.4, this exploration proved worthwhile.

For each criteria, any run for which the criteria's threshold is not breached, is discarded from the analysis of the performance of the criteria. In examining the usefulness of the stopping criteria when applied to these financial data sets, we are only concerned with the performance of the criteria *when a stopping generation is identified*. We therefore don't know if this would have been the optimal stopping point in the context of the run completing its 50 generations. The final column in each of Tables 8.2, 8.3 and 8.4, entitled 'Prob Halting', displays the proportion of all 50 runs in which a stopping generation was identified by the criterion in question (the values in the 'Prob Halting' column are scaled between 0 and 1). To calculate the number of runs each criterion acted upon, multiply the probability of halting by 50.

Table 8.3 Criteria Performance on Nikkei 255: Averages and (Standard Deviations) Over 50 Runs

Criterion	Slowness	Generalisability	Prob Best Crit	Prob Halting
GL_1	0.88 (0.98)	0.77 (0.6)	0	0.12
GL_2	0.93 (0.98)	0.77 (0.6)	0	0.12
GL_3	0.97 (0.99)	0.76 (0.6)	0	0.12
GL_5	0.99 (0.99)	0.76 (0.6)	0.02	0.12
$PQ_{0.5}$	8.41 (11.03)	0.95 (0.43)	0	0.98
$PQ_{0.75}$	8.59 (11.11)	0.94 (0.43)	0	0.98
PQ_1	8.89 (11.27)	0.96 (0.39)	0.02	0.98
PQ_2	9.71 (11.74)	0.95 (0.39)	0	0.98
PQ_3	10.57 (12.17)	0.95 (0.37)	0	0.98
UP_2	7.56 (10.75)	0.96 (0.25)	0.02	0.86
UP_3	5.44 (9.71)	0.96 (0.16)	0	0.32
UP_4	4.36 (5.26)	0.99 (0.02)	0	0.18
$COR_{0.8}$	7.42 (10.44)	0.91 (0.53)	0.32	1
$COR_{0.6}$	8.05 (10.72)	0.93 (0.47)	0.12	1
$COR_{0.4}$	8.86 (11.08)	0.95 (0.39)	0.08	0.98
$COR_{0.2}$	9.84 (11.56)	0.98 (0.29)	0.04	0.92
COR_0	10.69 (12.07)	0.99 (0.2)	0.04	0.86
$COR_{-0.2}$	10.07 (10.86)	0.98 (0.2)	0.02	0.78
$COR_{-0.4}$	9.69 (10.13)	0.98 (0.23)	0.08	0.72
$COR_{-0.8}$	9.14 (10.43)	0.99 (0.36)	0.24	0.54

Analysing Performance

Some notation, which will be used below, is defined here. The time at which training was stopped (the stopping generation) is denoted by t_s. The test fitness at the generation of the result which is produced by applying criterion C, is $E_{te}(C)$.

Prechelt [21] defines a criterion as being *good*, if it is among those that find the lowest validation set error for the entire run. The *best* criterion of each run, is defined as a good criterion which has an earlier stopping generation than other good criteria. That is, among all the criteria that find the best validation fitness, the *best* criterion of all is that which stopped training quickest [21].

We analysed the performance of the stopping criteria on each of the three benchmark financial data sets[1]. Three performance metrics were defined. The **'slowness'** of a criterion C in a run, relative to the best criterion x, is defined as

$$S_x(C) = t_s(C)/t_s(x).$$

The **"generalisability"** of a criterion C in a run, relative to the best criterion x, is defined [21] as

$$B_x(C) = E_{te}(C)/E_{te}(x).$$

Table 8.4 Criteria Performance on S&P500: Averages and (Standard Deviations) Over 50 Runs

Criterion	Slowness	Generalisability	Prob Best Crit	Prob Halting
GL_1	6.5 (8.19)	0.74 (0.33)	0	0.08
GL_2	6.5 (8.19)	0.74 (0.33)	0	0.08
GL_3	6.5 (8.19)	0.74 (0.33)	0	0.08
GL_5	6.5 (8.19)	0.74 (0.33)	0	0.08
$PQ_{0.5}$	7.79 (11.13)	0.79 (1.12)	0.02	1
$PQ_{0.75}$	8.15 (11.35)	0.76 (0.97)	0.02	1
PQ_1	8.52 (11.59)	0.76 (0.95)	0.02	1
PQ_2	9.45 (12.26)	0.74 (0.9)	0.02	0.98
PQ_3	10.13 (12.75)	0.74 (0.86)	0	0.98
UP_2	7.07 (10.21)	0.64 (0.8)	0	0.74
UP_3	9.24 (11.47)	0.59 (0.9)	0	0.44
UP_4	7 (5.95)	0.48 (1.02)	0	0.12
$COR_{0.8}$	7.21 (10.78)	0.76 (1.28)	0.18	1
$COR_{0.6}$	8.04 (11.24)	0.8 (1.25)	0.26	0.98
$COR_{0.4}$	8.76 (11.67)	0.79 (1.3)	0.04	0.96
$COR_{0.2}$	9.56 (12.07)	0.8 (1.3)	0.1	0.9
COR_0	10.58 (12.61)	0.92 (1.22)	0.04	0.88
$COR_{-0.2}$	10.76 (12.7)	0.96 (1.28)	0.08	0.88
$COR_{-0.4}$	10.98 (12.97)	0.97 (1.49)	0.02	0.82
$COR_{-0.8}$	2.24 (3.62)	0.68 (2.26)	0.2	0.64

The 'generalisability' of a criterion therefore measures how well the result it produces generalises beyond the training and validation sets, relative to the best criterion. Tables 8.2, 8.3 and 8.4 show the average values for slowness and generalisability over all the runs in which halting took place, for each criterion. Standard deviations are shown in brackets.

The third and final metric measures the **probability that a particular criterion will be the best criterion for a run**. This is calculated as follows. The number of times a particular criterion was chosen as the best criterion of a run is counted. This number is divided by the total number of runs, to give the average number of times the criterion is chosen as the best criterion. This average is used to measure the probability that the criterion will be the best criterion.

The correlation criteria come out very well in terms of having a high probability of being judged the best criterion of a run in Tables 8.2, 8.3 and 8.4. This is a very interesting and potentially useful result. For the S&P 500 dataset, the correlation criteria have better generalisation performance than all other criteria (averaging performance values across all criteria in a particular class). Their generalisation performance is marginally worse than the UP criteria for the Nikkei 255 dataset, but they outperform the other classes of criteria. They perform worse than all classes of criteria for the Euro/USD dataset, in terms of generalisation. The GL criteria are the fastest criteria at identifying a stopping point in two out of the three datasets, and second best in the Euro/USD dataset. The correlation criteria are always slower than both the GL and UP criteria, across all three datasets (once again, taking the average performance across all criteria for each class). The correlation criteria may still be a superior choice than the other three classes of criteria, unless training time is a constraint.

8.6 Conclusions

In this chapter, we focused on the techniques to counteract overfitting in evolutionary driven financial model induction, first providing some background to the problem. In order to clearly illustrate overfitting, we showed the existence of overfitting in evolved solutions to symbolic regression problems using grammar-based GP, using diagrams as visual aids. After noting other authors observations that traditional early stopping techniques, which usually stop training when validation fitness first disimproves, were often insufficient in increasing generalisation [21, 22], we applied four early stopping criteria to an exemplar trading rule induction problem using grammar-based GP. Three of these criteria had previously been implemented in the neural network literature [21]. The fourth class of criteria, which measured the correlation between the training and validation fitness at each generation and stopped training once the correlation dropped below a predefined threshold (such as 0.6), proved very successful. This class of criteria had a high probability of being a best criterion of a run, meaning that it would stop sooner than any other criteria after the global maximum validation set fitness had been observed. It also usually had adequate generalisation performance with respect to independent test data, that had neither been used for training, nor to determine stopping points.

As for the other classes of criteria, the UP criteria stopped after a persistent disimprovement had been observed in the validation fitness over consecutive time periods. UP criteria generalise well - they display the best results for generalisation on the Euro/USD and Nikkei 225 dataset. They are by no means slow to stop either - being fastest at identifying a stopping point on the Euro/USD dataset, and second fastest on both the Nikkei 225 and S & P 500 datasets. If a low training time is important, then the GL criteria are the best option - however, the UP criteria are a very attractive alternative, given they also display good generalisation performance.

Several areas are indicated for future work. In early stopping, the evolutionary process could be extended to include parameters such as the appropriate stopping threshold, or indeed to the evolution of high-quality stopping rules from low-level stopping criteria primitives (a meta-heuristic approach). We also intend to extend our work to embrace other generalisation methodologies from the statistical literature.

Acknowledgements. All of the authors acknowledge the financial support of Science Foundation Ireland (Grant No. 08/SRC/FMC1389) under which this research was conducted.

References

1. Agapitos, A., O'Neill, M., Brabazon, A.: Evolutionary Learning of Technical Trading Rules without Data-Mining Bias. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 294–303. Springer, Heidelberg (2010)
2. Banzhaf, W., Nordin, P., Keller, R., Francone, F.: Genetic programming: An introduction. Morgan Kaufmann, San Francisco (1999)
3. Becker, L.A., Seshadri, M.: Comprehensibility and overfitting avoidance in genetic programming for technical trading rules. Worcester Polytechnic Institute, Computer Science Technical Report (2003)
4. Brabazon, A., Dang, J., Dempsey, I., O'Neill, M., Edelman, D.: Natural computing in finance: a review. In: Handbook of Natural Computing: Theory, Springer, Heidelberg (2010)
5. Brabazon, A., O'Neill, M.: Biologically inspired algorithms for financial modelling. Springer, Heidelberg (2006)
6. Costelloe, D., Ryan, C.: On improving generalisation in genetic programming. In: Vanneschi, L., Gustafson, S., Moraglio, A., De Falco, I., Ebner, M. (eds.) EuroGP 2009. LNCS, vol. 5481, pp. 61–72. Springer, Heidelberg (2009)
7. Dempsey, I., O'Neill, M., Brabazon, A.: Foundations in Grammatical Evolution for Dynamic Environments. Springer, Heidelberg (2009)
8. Domingos, P.: The role of Occam's razor in knowledge discovery. Data Mining and Knowledge Discovery 3(4), 409–425 (1999)
9. Gagné, C., Schoenauer, M., Parizeau, M., Tomassini, M.: Genetic programming, validation sets, and parsimony pressure. In: Collet, P., Tomassini, M., Ebner, M., Gustafson, S., Ekárt, A. (eds.) EuroGP 2006. LNCS, vol. 3905, pp. 109–120. Springer, Heidelberg (2006)
10. Gencay, R., Qi, M.: Pricing and hedging derivative securities with neural networks: Bayesian regularization, early stopping, and bagging. IEEE Transactions on Neural Networks 12(4), 726–734 (2002)
11. Kushchu, I.: Genetic programming and evolutionary generalization. IEEE Transactions on Evolutionary Computation 6(5), 431–442 (2002)
12. Luke, S., Panait, L.: Lexicographic parsimony pressure. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 829–836. Morgan Kaufmann, San Francisco (2002)
13. Luke, S., Panait, L.: A comparison of bloat control methods for genetic programming. IEEE Transactions on Evolutionary Computation 14(3), 309–344 (2006)
14. Mckay, R.I., Hoai, N.X., Whigham, P.A., Shan, Y., O'Neill, M.: Grammar-based Genetic Programming: a survey. Genetic Programming and Evolvable Machines 11(3-4), 365–396 (2010)

15. Mitchell, T.: Machine Learning. McGraw-Hill, New York (1997)
16. O'Neill, M., Hemberg, E., Gilligan, C., Bartley, E., McDermott, J., Brabazon, A.: GEVA: grammatical evolution in Java. ACM SIGEVOlution 3(2), 17–22 (2008)
17. O'Neill, M., Ryan, C.: Grammatical Evolution: Evolutionary automatic programming in an arbitrary language. Kluwer, Dordrecht (2003)
18. O'Neill, M., Vanneschi, L., Gustafson, S., Banzhaf, W.: Open issues in genetic programming. Genetic Programming and Evolvable Machines 11(3-4), 339–363 (2010)
19. Paris, G., Robilliard, D., Fonlupt, C.: Exploring overfitting in genetic programming. In: Liardet, P., Collet, P., Fonlupt, C., Lutton, E., Schoenauer, M. (eds.) EA 2003. LNCS, vol. 2936, pp. 267–277. Springer, Heidelberg (2004)
20. Poli, R., Langdon, W.B., McPhee, N.F.: A field guide to genetic programming. Lulu Enterprises, UK (2008)
21. Prechelt, L.: Early stopping - but when? In: Orr, G.B., Müller, K.-R. (eds.) NIPS-WS 1996. LNCS, vol. 1524, p. 55. Springer, Heidelberg (1998)
22. Thomas, J.D., Sycara, K.: The importance of simplicity and validation in genetic programming for data mining in financial data. In: Proceedings of the Joint GECCO 1999 and AAAI 1999 Workshop on Data Mining with Evolutionary Algorithms: Research Directions, pp. 7–11 (1999)
23. Tuite, C., Agapitos, A., O'Neill, M., Brabazon, A.: A Preliminary Investigation of Overfitting in Evolutionary Driven Model Induction: Implications for Financial Modelling. In: Di Chio, C., Brabazon, A., Di Caro, G.A., Drechsler, R., Farooq, M., Grahl, J., Greenfield, G., Prins, C., Romero, J., Squillero, G., Tarantino, E., Tettamanzi, A.G.B., Urquhart, N., Uyar, A.Ş. (eds.) EvoApplications 2011, Part II. LNCS, vol. 6625, pp. 120–130. Springer, Heidelberg (2011)
24. Tuite, C., Agapitos, A., O'Neill, M., Brabazon, A.: Early Stopping Criteria to Counteract Overfitting in Genetic Programming. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation. ACM, New York (2011) (forthcoming)
25. Vanneschi, L., Gustafson, S.: Using crossover based similarity measure to improve genetic programming generalization ability. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, pp. 1139–1146. ACM, New York (2009)

9

An Order-Driven Agent-Based Artificial Stock Market to Analyze Liquidity Costs of Market Orders in the Taiwan Stock Market

Yi-Ping Huang[1], Shu-Heng Chen[2], Min-Chin Hung[1], and Tina Yu[3]

[1] Department of Financial Engineering and Actuarial Mathematics, Soochow University, Taipei, Taiwan
berthuang@cht.com.tw, nhungg@scu.edu.tw
[2] AI-ECON Research Center, Department of Economics, National Chengchi University, Taipei, Taiwan
chen.shuheng@gmail.com
[3] Department of Computer Science, Memorial University of Newfoundland, St John's, NL A1B 3X5, Canada
gwoingyu@gmail.com

Summary. We developed an order-driven agent-based artificial stock market to analyze the liquidity costs of market orders in the Taiwan Stock Market (TWSE). The agent-based stock market was based on the DFGIS model proposed by Daniels, Farmer, Gillemot, Iori and Smith [2]. When tested on 10 stocks and securities in the market, the model-simulated liquidity costs were higher than those of the TWSE data. We identified some possible factors that have contributed to this result: 1) the overestimated effective market order size; 2) the random market orders arrival time designed in the DFGIS model; and 3) the zero-intelligence of the artificial agents in our model. We continued improving the model so that it could be used to study liquidity costs and to devise liquidation strategies for stocks and securities traded in the Taiwan Stock Market.

9.1 Introduction

Market liquidity, or the ability of an asset to be sold without causing a significant amount of price movement and with minimum loss of value, plays an important role in financial investment and in securities trading. One recent event that highlighted the impact of asset liquidity on financial institutions was the collapse of Bear Stearns during the subprime crises. Bear Stearns was involved in securitization and issued a large amount of asset-backed securities, mostly mortgage-backed assets, which fell dramatically in liquidity and value during the crisis. In March 2008, the Federal Reserve Bank of New York provided an emergency loan to try to avert the collapse of the company. However, the company could not be saved and was subsequently sold to JP Morgan Chase in 2008.

In large investment institutions, the liquidation of a large block of assets within a given time constraint to obtain cash flow arises frequently. For example, a financial institution may

A. Brabazon et al. (Eds.): Natural Computing in Comput. Finance, SCI 380, pp. 163–179.
springerlink.com

need to liquidate part of its portfolio to pay for its immediate cash obligations. One possible liquidation strategy is to sell the entire block of assets at once. However, this high-volume trading can cause the price of the share to drop between the time the trade is decided and the time the trade is completed. This implicit cost (due to the price decline) is known as the market impact cost (MIC) or liquidity cost (the numerical definition is given in Section 9.3). To minimize such cost, a better strategy is to divide the block of assets into chunks and sell them one chunk at a time. However, in what way should those chunks be sold so that the liquidity cost is minimized?

In Algorithmic Trading, where computer programs are used to perform asset trading including deciding the timing, price, or the volume of a trading order, this liquidation problem is characterized as an optimization problem. With a smooth and differentiable utility function, the problem can be solved mathematically [1, 6].

However, this mathematical approach to find an optimal liquidation strategy has some shortcomings, such as the imposed assumption that price dynamics are an arithmetic random walk with independent increments. In this chapter, we adopt a different approach by devising an agent-based artificial stock market, which is empirically driven and simulation-based (the model is explained in Section 9.3). It, therefore, can in principle work with various stochastic processes where analytical solutions would be difficult to derive. By performing simulations and analyzing liquidity costs induced under different market scenarios, we hope to understand the dynamics of liquidity costs, and hence to devise a more realistic optimal liquidation strategy.

The rest of this chapter is organized as follows. In Section 9.2, we provide the background and summarize related works. Section 9.3 explains the agent-based artificial stock market we developed based on the DFGIS model and based on data from the Taiwan Stock Market (TWSE). In Section 9.4, the 10 securities and stocks that we selected to conduct our study are presented. Section 9.5 provides the model parameters used to perform our simulation. We analyze the simulation results in Section 9.6 and present our discussions in Section 9.7. Finally, Section 9.8 concludes the chapter with an outline of our future work.

9.2 Background and Related Work

This study implements an agent-based model for an order-driven double auction market, which is the most common financial market in the world. We shall first give a brief introduction to the basic microstructure and trading mechanism of a standard order-driven double-auction market. Next, the DFGIS model [2] on which our agent-based artificial stock market is based will be presented. After that, we will summarize the work of [5] on agent-based models used to study liquidation strategies at the end of the section.

9.2.1 Order-Driven Double-Auction Markets

In an order-driven double-auction market, prices are determined by the publication of *orders* to buy or sell shares. This is different from a quote-driven market where prices are determined from *quotations* made by market makers or dealers.

There are two basic kinds of orders in an order-driven market. Impatient traders submit *market orders*, which are requests to buy or sell a given number of shares immediately at the best available price. More patient traders submit *limit orders*, which specify the limit (best acceptable) price for a transaction. Since limit orders often fail to result in immediate transactions, they are stored in a *limit order book*. As shown on the left of Table 9.1, limit

Table 9.1 An example of an order book before (left) & after (right) a transaction

Limit Buy Orders		Limit Sell Orders	
Size	Price	Size	Price
20	\$1.10	35	\$1.17
25	\$1.09	10	\$1.19
35	\$1.05	20	\$1.21
10	\$1.01	15	\$1.25

Limit Buy Orders		Limit Sell Orders	
Size	Price	Size	Price
15	\$1.09	35	\$1.17
35	\$1.05	10	\$1.19
10	\$1.01	20	\$1.21
		15	\$1.25

buy orders are stored in decreasing order of limit prices while limit sell orders are stored in increasing order of limit prices. The buy limit orders are called *bids* and the sell limit orders are called *asks*. For a normal double-auction market, the best (highest) bid price is lower than the best (lowest) ask price. The difference between the two is called the *spread* of the market. In the example in Table 9.1 (left), the *spread* is \$0.07.

When a *market order* arrives, it is matched against limit orders on the opposite side of the book. For example, when a market sell order for 30 shares arrives at the market whose order book is as that on the left of Table 9.1, it will first be matched against the current best bid (20 shares at \$1.10 per share). Since the size of the sell order (30 shares) is larger than that of the best bid (20 shares), the remainder of the market sell order (10 shares) will be matched against the next best bid (25 shares at \$1.09 per share). After the transaction is completed, the limit order book will change to the right of Table 9.1 and the market spread will widen to \$0.08.

9.2.2 The DFGIS Model

In the original DFGIS model [2], all the order flows (including limit orders and market orders) are modeled as a Poisson process. Market orders arrive at the market in chunks of σ shares (where σ is a fixed integer) at an average rate of μ per unit of time. A market order may either be a buy market order or a sell market order with equal probability.

Limit orders arrive at the market in chunks of σ shares, at an average rate of α shares per unit price per unit of time. A limit order may either be a limit buy order or a limit sell order with equal probability. The limit prices in limit orders are generated randomly from a uniform distribution. In particular, the limit sell prices have a range between $(-\infty, a(t))$, where $a(t)$ is the best (lowest) *ask* price in the market at time t. Similarly, the limit buy prices have a range between $(b(t), \infty)$, where $b(t)$ is the best (highest) *bid* in the market at time t. The prices are first converted to logarithms so that the value is always positive. In addition, the price changes are not continuous, but have discrete quanta called *ticks* (represented as dp). Tick size is the price increment/decrement amount allowed in a limit order.

DFGIS also allows the limit order to expire or to be cancelled after being placed in the market. Limit orders are expired and cancelled according to a Poisson process, analogous to radioactive decay, with a fixed-rate δ per unit of time. Table 9.2 lists the parameters of the DFGIS model.

To keep the model simple, the DFGIS does not explicitly allow limit orders whose prices cross the best bid price or the best ask price. In other words, the price of a limit buy order must be below the best ask price and the price of a limit sell order must be above the best bid price. Farmer, Patelli and Zovko [4] implemented the model to explicitly handle this type of order. They defined *effective market orders* as shares that result in transactions immediately and *effective limit orders* as shares that remain on the order book. A limit order with a price

Table 9.2 Summary of DFGIS parameters

Parameter	Description	Dimension
α	avg. limit order rate	share/price · time
μ	avg. market order rate	share/time
δ	avg. limit order decay rate	1/time
σ	order size	a constant share
dp	tick size	price

Table 9.3 An example of an order book before (left) & after (right) handling a limit sell order with price equal to the best bid

Limit Buy Orders		Limit Sell Orders	
Size	Price	Size	Price
20	\$1.10	35	\$1.17
25	\$1.09	10	\$1.19
35	\$1.05	20	\$1.21
10	\$1.01		

Limit Buy Orders		Limit Sell Orders	
Size	Price	Size	Price
25	\$1.09	10	\$1.10
35	\$1.05	35	\$1.17
10	\$1.01	10	\$1.19
		20	\$1.21

that crosses the opposite best price is split into effective market orders and effective limit orders according to the above definition. For example, when a limit sell order of 30 shares with price \$1.10 arrives at the market (with the order book as that listed in Table 9.3 (left)), the order will be split into an effective market order of 20 shares and an effective limit sell order of 10 shares. After the execution of the 20 shares of the effective market order (at price \$1.10), the order book is changed to that on the right of Table 9.3.

9.2.3 The Guo Agent-Based Stock Market Model

Guo [5] implemented an agent-based artificial stock market based on the DFGIS model (he called it the SFGK model) to study time-constrained asset liquidation strategies through market sell orders only. In particular, he compared the performance of two strategies. The first one uniformly divides the liquidation shares X and the time constraint T into N chunks. This "uniform rhythm" strategy instructs a trader to sell X/N shares every T/N seconds, regardless of the market condition.

The second strategy is the "non-uniform rhythm" strategy which also divides the liquidation shares and time uniformly. However, within each time segment, this strategy requires a trader to continuously observe the market spread and initiates the selling of the X/N shares as soon as the current market spread, for the first time within the time segment, falls below the pre-determined spread threshold. If the market spread never falls below the spread threshold for a time segment, the strategy will involve selling the X/N shares at the end of the time segment.

Guo devised one agent (A) with the "uniform rhythm" strategy and another agent (B) with the "non-uniform rhythm" strategy. He tested each agent individually by running 200 simulations independently. For each simulation run, the number of liquidation shares (X) is 20 and the time constraint (T) is 5 minutes. The total asset is divided into 10 chunks, each of which contains 2 shares that will be sold within a time segment of 30 seconds. For each market order the agent submits, the number of shares (α) is 1.

The performance of the two agents is evaluated by the *average selling price per share* relative to the *volume weighted average price in the market*. In other words, it is the measure of how much better or worse an agent performs, when tested compared to the market. His simulation results indicated that agent *B* (based on the "non-uniform rhythm" strategy) outperformed the market while agent *A* (based on the "uniform rhythm" strategy) underperformed the market.

Guo did not explicitly study liquidity costs when devising his liquidation strategy. By contrast, we are interested in quantifying the cost in a real-world stock market. We therefore used TWSE stock order and transaction data (see Section 9.4) to construct the agent-based artificial stock market using the DFGIS model. We describe this agent-based system in the following section.

9.3 An Agent-Based Taiwan Stock Market Model

The agent-based artificial stock market consists of zero-intelligent agents who place *buy*, *sell* or *cancellation* orders at random, subject to the constraints imposed by the current prices. The distribution of the order prices and quantities and the distribution of the time intervals of the order submissions in the market follow that of the DFGIS model. Unlike the Algorithmic Trading model, which imposes unrealistic assumptions, the agent-based model is governed by the 5 DFGIS model parameters. The market properties emerge from the stochastic simulation. We describe our implementation of the abstract DFGIS model for the TWSE in the following subsections.

9.3.1 Buy and Sell Orders

On the TWSE, a submitted order (either buy or sell) needs to specify the price, in addition to the quantity, that the trader is willing to accept. However, the price can cross the *disclosed best prices* (explained in Section 9.3.3). When the price of a buy order is greater than or equal to the disclosed best ask price or the price of a sell order is less than or equal to the disclosed best bid price, there is a match for the transaction. We follow [4] (see Section 9.2.2) and call the portion of an order that results in an immediate transaction an *effective market order*. The non-transacted portion of a submitted order is recorded on the order book and is called the *effective limit order*.

9.3.2 Event-Time Model

We implemented the artificial stock market as an event-time model, where the events in the model are not connected to real time. We first partitioned a trading day into a fixed number of time intervals. The events taking place at each time interval become the event-time series describing the market activities of that day.

The TWSE opens at 9:00 am and closes at 1:30 pm. With a time interval of 0.01 seconds, the event-time series of a trading day is 1,620,000 time intervals long. The TWSE is a *call auction* market where the submitted orders are matched once every 25 seconds. Hence, there are 648 order-matching events in a daily even-time series and the number of events between two order-matching events is 2,499. There are five possible events in an event-time series:

- Effective limit order submission: This has an average rate α and is denoted by `L`.
- Effective market order submission: This has an average rate μ and is denoted by `M`.

- Order cancellation submission: This has an average rate δ and is denoted by `C`.
- Order matching: The buy and sell orders are matched for transactions and is denoted by `T`.
- No activity: None of the above events occurred in the market and is denoted by `N`.

From a simulation point of view, the simulation result based on an event-time series and that based on a real-time series are equivalent. For example, the event-time series `LNNNM` is equivalent to the real-time series `L`... (real time elapse)... `M`. However, the event-time model is easier to implement and has a shorter simulation running time because there is no need to handle the time elapse between two events. We therefore adopted the event-time model to implement our artificial stock market.

9.3.3 Order Pricing Rules

As mentioned previously, the TWSE matches submitted orders once every 25 seconds. After a match is completed, some of the orders become effective market orders while some become effective limit orders. The five best prices are then disclosed to the public. During the following 25 seconds of the waiting-for-matching period, the TWSE does not disclose any information about the newly-submitted orders. Consequently, investors do not have the updated best prices, but rather the best prices from the previous matching period to make trading decisions. These *disclosed best prices* are then used to decide order pricing ranges and to calculate the liquidity costs (explained in the next subsection).

The TWSE has a pricing rule whereby the price range of an order has to be between the closing price on the previous trading day (`cp`) $\pm$ 7%. Thus, the submitted orders in our simulation system have the following price ranges:

- Effective limit buy orders: (`cp`-7%, `da(t)`), where `da(t)` is the disclosed best (lowest) *ask* price in the market at time t,
- Effective limit sell orders: (`db(t)`, `cp`+7%), where `db(t)` is the disclosed best (highest) *bid* in the market at time t.
- Effective market buy orders: `cp`+7%. This guarantees an immediate transaction.
- Effective market sell orders: `cp`-7%. This guarantees an immediate transaction.

The tick sizes (the price increment/decrement amount) are as defined by the TWSE.[1]

9.3.4 Liquidity Costs

We define *liquidity cost* as the difference between the *expected transaction payment* and the *actual transaction payment* of an effective market order. The expected transaction payment is calculated by multiplying the disclosed best price by the number of shares of an effective market order. Since the disclosed best price is from the previous transaction period, this is the expected amount of payment to be made during the current matching period. The actual transaction payment, however, can be different from what was expected. It is calculated as the *executed transaction price* (explained in the next paragraph) multiplied by the the number of shares in a transaction. We can interpret liquidity cost as the difference between an investor's expected transaction payment and the actual transaction payment he/she made.

The TWSE uses a special rule to decide the execution transaction price to match the submitted orders. Instead of the best ask and the best bid in the current matching period, the

[1] See `http://www.twse.com.tw/ch/trading/introduce/introduce.php#1`

Table 9.4 An example of the determination of the execution transaction price

accumulated buy shares	buy shares	price	sell shares	accumulated sell shares	transaction volume (shares)
99	99	102.5	1	103	99
99		102	2	102	99
121	22	101.5	99	100	100
126	5	100	1	1	1

rule selects the price that gives the maximum transaction volume as the execution transaction price. Table 9.4 gives an example of how the transaction price is decided.

In column 2, the buy order quantities under the prices in column 3 are given. Column 4 gives the sell order quantities under the prices in column 3. As an example, the first row shows that there is a buy order bidding \$102.5 for 99 shares and there is a sell order asking for the same price for 1 share. Column 6 gives the number of transaction shares under the price in column 3. In this case, \$101.5 gives the largest transaction volume (100 shares) and is selected as the execution transition price for this matching period.

With the selected execution transaction price (TP) and the disclosed best ask (DBA), the liquidity cost of an *effective market buy order* with volume V is defined as:

$$LC_{buy} = \frac{TP \times V - DBA \times V}{DBA \times V} \tag{9.1}$$

Similarly, with the disclosed best bid (DBB), the liquidity cost of an *effective market sell order* is defined as:

$$LC_{sell} = \frac{DBB \times V - TP \times V}{DBB \times V} \tag{9.2}$$

We normalized the liquidity cost such that the value is the ratio of the original cost to the expected transaction payment.

Both LC_{buy} and LC_{sell} can be positive or negative. This is because, under the TWSE pricing regulation, the execution transaction price may be higher or lower than what a trader has expected. Liquidity cost with a negative value means that the execution transition price is better than what the trader has expected while liquidity cost with a positive value means that the opposite situation applies.

When an order is partially or not executed in the current matching period, the non-executed portion remains in the order book. These effective limit orders may later be executed and become effective market orders. However, effective limit orders may lead to a loss of opportunities related to the changing market prices or a decaying value of the information responsible for the original trading decision. This so-called *opportunity cost* is difficult to estimate, and hence is not considered in our liquidity costs calculation.

9.3.5 Measurement of Model Parameters

The model parameters are estimated using real data from the TWSE (see Section 9.4). These parameters include an effective market order rate (μ), effective limit order rate (α),

Table 9.5 Model Parameter Estimation

Parameter	Description	Value	Dimension
$p(\mu)$	avg. daily effective market order event rate	$\sum_{t=1}^{n} p_t(\mu) \times w_t$	order-event/time
$p(\alpha)$	avg. daily effective limit order event rate	$\sum_{t=1}^{n} p_t(\alpha) \times w_t$	order-event/time
$p(\delta)$	avg. daily cancellation order event rate	$\sum_{t=1}^{n} p_t(\delta) \times w_t$	order-event/time
σ	avg. order size	$\sum_{t=1}^{n} \sigma_t \times w_t$	share/order

cancellation order rate (δ), and order size (σ). Note that tick size (dp) has been discussed previously in Section 9.3.3.

For each parameter, we calculated the mean of the daily value weighted by the number of daily events. For example, the parameter $p_t(\mu)$ is the ratio of the number of effective market order events (including buy and sell orders) to the total number of buy, sell, cancellation orders and non-active events (1,619,352) on day t. The weight factor w_t is the ratio of the number of order events (including effective market, effective limit and cancellation orders) on day t to the total number of order events for the entire period:

$$w_t = \frac{n_{\mu t} + n_{\alpha t} + n_{\delta t}}{\sum_{i=1}^{n} n_{\mu i} + n_{\alpha i} + n_{\delta i}} \tag{9.3}$$

where $n_{\mu t}$ is the number of effective market orders on day t; $n_{\alpha t}$ is the number of effective limit orders on day t; $n_{\delta t}$ is the number of cancellation orders on day t; and n is the number of days in the entire period. We measured $w_t \times p_t(\mu)$ across the whole period (77 trading days in this case) and then added them together, which becomes the average daily effective market order event rate $p(\mu)$. Note that its dimension is `order-event/time`, which is slightly different from μ of [2] whose dimension is `share/time` (see Table 9.2). We applied the same method to calculate $p(\alpha)$ and $p(\delta)$. With that, we can calculate the average daily no-activity event rate ($p(n)$) as $1 - p(\mu) - p(\alpha) - p(\delta)$.

To calculate the average order size, we first computed the average number of shares in the effective market and effective limit orders submitted on day t (σ_t) (excluding those submitted before the first best prices were disclosed and after the market was closed). The summation of $\sigma_t \times w_t$ for the entire period becomes the average order size σ.

In the simulation, we used a stochastic order size, which is generated randomly from a half-normal distribution with standard deviation $\sqrt{\frac{\pi-2}{2}} \times \sigma$ (σ is the average order size, not standard deviation)[7]. According to [2], this stochastic order size gives the same result as that produced from the constant order size σ. According to the TWSE regulation, the maximum order size is 499,000 shares. Table 9.5 summarizes the model parameters implemented in our system, where n stands for the number of trading days (77) in the data set. Note that this approach to the estimation of the model parameters is similar to that of [4]. It assumes that these parameters are time-invariant, which is an important assumption in this study.

9.3.6 Program Implementation and System Flow

The simulation program was implemented in the Python programming language. Figure 9.1 depicts the overall system workflow. Each simulation is for one trading day for the TWSE.

Initially a series of events on a trading day is generated, based on $p(\mu)$, $p(\sigma)$, $p(\delta)$ and $p(n)$. The number of events is 1,620,000. These events are then executed sequentially, according to what types of events they are.

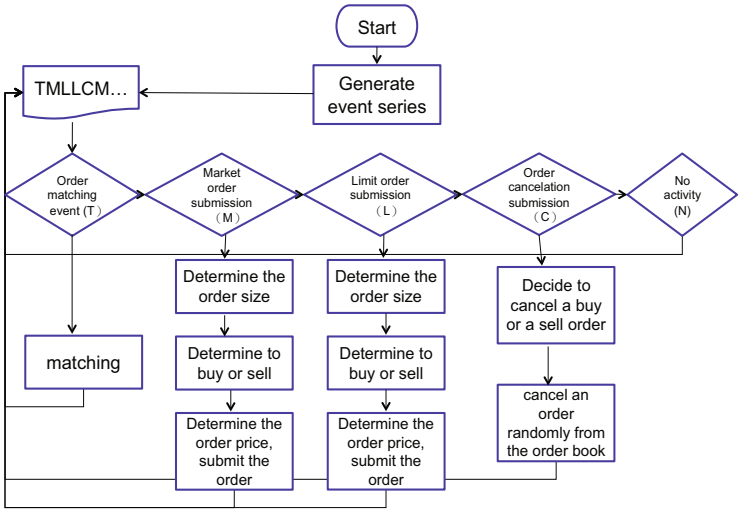


Fig. 9.1 The overall system workflow

If the event is an order matching event (T), the program matches orders and carries out transactions. If it is an effective market order submission (M) or an effective limit order submission event (L), the program decides the order size based on the half-normal distribution of α. Next, the program decides if it is a buy or a sell order with an equal probability (50%). After that, the order price is determined by a *uniformly* random draw from the range given in Section 9.3.3 as what the zero-intelligence agent is supposed to do.[2] The order is then submitted.

If it is an order cancellation submission event (C), the program decides whether to cancell a buy or a sell order with equal probability (50%). Next, an order on the order book is cancelled randomly. If it is an non-activity event (N), the program continues to process the next event.

9.4 The Data Set

In recent years, the types of securities traded in a stock market have increased from stocks and warrants to Exchange Traded Funds (ETFs) and Real Estate Investment Trust Funds (REITs). ETFs are baskets of stocks which are vehicles for passive investors who are interested in long-term appreciation and limited maintenance. A REIT is a popular investment option as it has better liquidity, in theory, than real estate. We therefore selected a variety of stocks, ETFs and REITs traded in the TWSE to study liquidity costs. They are selected to cover a wide variety of characteristics (see Table 9.6).

[2] Here, we simply follow [4] to operate the zero-intelligence agent.

Table 9.6 The 10 selected securities and their characteristics

Security	Ticker	Characteristics
Taiwan Top50 Tracker Fund	0050	ETF with the highest trading volume
Polaris/P-shares Taiwan Dividend+ ETF	0056	ETF with a low trading volume
Cathay No.2 Real Estate Investment Trust	01007T	REIT with a high trading volume
Gallop No.1 Real Estate Investment Trust Fund	01008T	REIT with a low trading volume
China Steel	2002	Blue chip stock in TWSE
TSMC	2330	Stock with a high trading volume and the largest market capitalization
MediaTek	2454	Stock with a high unit price
HTC	2498	Stock with a high unit price
President Chain Store	2912	Stock with a large market capitalization but a low trading volume
Inotera	3474	Non-blue chip stock on the TWSE (there is a net loss during the fiscal year)

The data provided by the TWSE include daily order data, transaction data and disclosed price data from February 2008 to May 2008 (77 trading days). Based on Equation 9.1 and 9.2, we calculated the liquidity costs of all effective market order transactions for the 10 securities. We neglect the effective market orders that arrive before the first best price is disclosed, since the opposing best prices of these orders are not available. The same applies to the effective market orders that arrive after the market is closed. These orders would be recorded on the order book and executed the following day (or later) if their prices have a match.

We are particularly interested in the liquidity costs of effective market orders that were traded immediately right after the orders were submitted, as they were a strong indication of the market liquidity of a security. The liquidity costs for orders, whose transactions took place after the orders have been entered (and waited for) in the order book, might be influenced by other factors, such as opportunity cost, and hence are less informative about the liquidity of a security.

Table 9.7 gives the liquidity cost statistics of the effective market orders with immediate transactions that were executed during the 77 trading days. It shows that most securities have negative average liquidity costs, except the two securities that have the lowest transaction frequency (0100T and 01008T).[3] The maximum amount of liquidity cost for a transaction is less than 3%. This indicates that these securities have high market liquidity.

We noted that China Steel (2002.TW) and TSMC (2330.TW) have the highest trading frequency (472,354 & 483,565). This might be due to the fact that 2002.TW is a blue chip stock while 2330.TW has a high market capitalization. They are attractive to domestic and foreign investors who seek secure and stable returns.

[3] The results of negative liquidity costs may be a little intriguing, and need to be addressed in further study.

Table 9.7 Descriptive statistics of the liquidity costs of effective market orders with immediate transactions, based on the TWSE data

Ticker	Max	Min	Mean	Std. Dev.	Kurtosis	Sum Sq. Dev	No. of Transactions
0050	0.69%	-0.92%	-0.02%	0.000639	23.17	0.04	101419
0056	2.22%	-1.4%	-0.01%	0.000743	122.77	0.01	16040
01007T	2.55%	-1.58%	0%	0.00091	252.81	0	4912
01008T	1.12%	-1.68%	0.01%	0.001133	90.85	0	872
2002	1.7%	-1.89%	-0.05%	0.001119	52.72	0.59	472354
2330	1.73%	-1.88%	-0.06%	0.001372	48.72	0.91	483565
2454	1.33%	-1.47%	-0.04%	0.001525	20.11	0.95	409985
2498	1.5%	-1.63%	-0.04%	0.001297	26.46	0.4	237582
2912	2.21%	-2.67%	-0.07%	0.00239	24.12	0.31	54480
3474	2.13%	-2.97%	-0.05%	0.002128	50.34	0.37	81450

Table 9.8 Ratio of the transaction volume of the effective market orders with immediate transactions to the total transaction volume of all orders, based on the TWSE data

LC range	(-3%,-2%]		(-2%,-1%]		(-1%,0%]		(0%,1%]		(1%,2%]		(2%,3%]	
Ticker	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max
0050	-	-	-	-	45.52%	65.88%	6.15%	24.51%	-	-	-	-
0056	-	-	0.01%	0.71%	40.88%	79.82%	9.22%	52.45%	0.01%	1%	0.05%	0.05%
01007T	-	-	0%	0.12%	44.7%	98.79%	4.17%	50.83%	0.01%	2.22%	0.04%	0.04%
01008T	-	-	0.05%	7.35%	45.39%	100%	2.59%	71.43%	0.02%	3.45%	-	-
2002	-	-	0.14%	12.74%	50.54%	71.49%	4.45%	26.33%	0.21%	18.4%	-	-
2330	-	-	0.46%	17.1%	47.38%	69.5%	4.07%	33.37%	0.43%	24.09%	-	-
2454	-	-	0.09%	4.56%	51.34%	68.51%	7.75%	21.25%	0.14%	8.45%	-	-
2498	-	-	0.02%	1.51%	48.62%	67.43%	5.59%	18.9%	0.05%	5.81%	-	-
2912	0.09%	8.9%	0.14%	6.54%	44.18%	77.66%	2.7%	21.82%	0.18%	11.41%	0.01%	0.01%
3474	0.12%	13.68%	0.17%	8.91%	45.54%	67.54%	7.03%	50.54%	0.51%	17.13%	0.35%	0.35%

To further analyze the market liquidity of these 10 securities, we partitioned the liquidity costs into 6 different value ranges. After that, we computed the ratio of the transaction volume of the effective market orders with immediate transactions to the total transaction volume of all orders (which include limit orders, not immediately executed market orders and so on). As shown in Table 9.8, the trading volume of this type of effective market order is more than 50% of the total transaction volume for all 10 securities. Meanwhile, the majority (40-50%) of all transaction volumes for these securities have their liquidity costs between -1% and 0% (see the 6th column of Table 9.8). These statistics further support them as high market liquidity securities.

9.5 Experimental Setup

Based on the definition in Table 9.5, we calculated $p(\mu)$, $p(\alpha)$, $p(\delta)$, $p(n)$ and σ for the 10 securities in Table 9.9. Although TSMC (2330.TW) is the largest market capitalization stock and is traded frequently, the average daily order size is 14,000 shares, which is much lower than that of the Taiwan Top50 Tracker Fund (0050.TW) (45,000 shares). This might be because the Taiwan Top50 Tracker Fund (0050.TW) is mostly traded by market makers,

who normally trade orders with a large volume, while TSMC (2330.TW) is traded by many different kinds of investors. For each of the securities, we made 10 simulation runs, each of which simulates one trading day for the TWSE. The simulation results are presented and analyzed in the following section.

Table 9.9 Computed $p(\alpha)$, $p(\mu)$, $p(\delta)$, $p(n)$ and σ for the 10 studied securities

Ticker	p(α) $\times 10^{-3}$	p(μ) $\times 10^{-3}$	p(δ) $\times 10^{-3}$	p(n) $\times 10^{-3}$	σ $\times 10^{3}$
0050	2.498	1.029	1.586	994.887	45
0056	0.404	0.147	0.232	999.217	19
01007T	0.109	0.059	0.042	999.79	21
01008T	0.026	0.01	0.003	999.961	22
2002	6.292	5.007	1.923	986.778	10
2330	6.676	5.059	2.218	986.047	14
2454	4.401	4.404	2.086	989.109	3
2498	2.57	2.404	1.182	993.844	3
2912	0.9	0.644	0.414	998.042	5
3474	1.195	0.96	0.459	997.386	11

9.6 Simulation Results and Analysis

Using the 10 days of simulation data, we calculated the daily average trading volume and the daily average number of transactions of the effective market orders. We then compared them with that calculated from the 77 days of TWSE data. As shown in Table 9.10, the results calculated from the simulation data are higher than those calculated from the TWSE data for almost all of the securities. This might be because, in our system, a crossing order could be split into effective market orders and effective limit orders (see Section 9.3.1). In most cases, the effective market order volume (shares) is smaller than the volume (shares) of the effective limit order. However, the DFGIS model assumes that the effective market order size is the same as the effective limit order size σ, which is calculated as the average size of both types of orders. In other words, the average order rate for effective market orders is overestimated. Consequently, the effective market order volume simulated based on σ is higher than that for the real TWSE data. One way to address this issue is to use two different order size parameters, one for effective market orders and one for effective limit orders, to conduct our simulation. We will investigate this option in our future work.

We also evaluated the liquidity cost statistics of effective market orders with immediate transactions, based on the simulation data (see Table 9.11). Compared to Table 9.7, the liquidity costs generated by the simulation data are higher than those for the TWSE data. This might also be due to the overestimated effective market order size σ in our system. With a higher effective market order size, the liquidity costs of effective market orders with immediate transactions are likely to be higher.

To rigorously evaluate the similarity between the simulated liquidity costs and the TWSE data, we performed the Mann-Whitney-Wilcoxon (MWW) test on all 10 securities. The resulting p-values are 0 across all 10 securities, indicating they are indeed different from one another.

Table 9.10 Daily effective market order trading volumes and the number of transactions comparison

Ticker	TWSE data		Simulation Data	
	daily trading volume (share)	daily no. of transactions	daily trading volume (share)	daily no. of transactions
0050	15411781	2554	73289200	2997
0056	1438702	352	5334100	506
01007T	1731457	144	2136900	190
01008T	332438	24	447200	34
2002	50360139	12062	64442500	11826
2330	69341433	12582	95335300	12765
2454	12749066	8010	17604900	7809
2498	6475480	4332	10357200	4788
2912	3228883	1369	5120400	1819
3474	11124524	2183	15510000	2794

Table 9.11 Descriptive statistics of the liquidity costs of effective market orders with immediate transactions, based on the simulation data

Ticker	Max	Min	Mean	Std. Dev.	Kurtosis	Sum Sq. Dev	No. of Transactions
0050	2.66%	-3.05%	-0.12%	0.006948	4.43	0.79	16470
0056	4.48%	-4.41%	0.18%	0.008189	9.38	0.16	2359
01007T	9.65%	-6.42%	0.5%	0.01279	14.84	0.15	889
01008T	6.39%	-1.94%	0.74%	0.012946	8.15	0.02	129
2002	1.79%	-2.22%	-0.18%	0.00562	3.42	2.56	81047
2330	2.03%	-2.35%	-0.21%	0.006375	3.37	3.31	81490
2454	2.93%	-3.62%	-0.28%	0.008793	3.45	5.51	71213
2498	3.76%	-4.65%	-0.3%	0.011251	3.78	4.9	38727
2912	7.32%	-11.11%	-0.11%	0.015314	8.4	2.42	10320
3474	5.35%	-6.43%	-0.28%	0.015003	4.38	3.5	15556

Similar to Table 9.8, we computed the ratio of the transaction volume of the effective market orders with immediate transactions to the total transaction volume for all orders under liquidity cost for 6 different ranges. However, the value ranges are partitioned slightly differently from those of Table 9.8. This is because the liquidity costs of the simulation data have a wider spread (-11.11%, 9.65%) than those of the TWSE data. Since we are more interested in positive liquidity costs, which are indicators of poor market liquidity, we grouped the negative liquidity costs into one bin and added two bins for liquidity costs beyond 3%. The results are given in Table 9.12.

As shown, the liquidity cost upper bound of 2002 (China Steel) and 2330 (TSMC) is 2%, which is the same for both simulation and TWSE data. Meanwhile, the two securities have higher liquidity cost transactions ratios (1% - 2%) for simulation and TWSE data that are similar to each other (the difference is $\sim 1\%$). Similarly, the negative liquidity cost transaction ratios ($12\% - 0\%$) for the simulation and TWSE data these two securities are not too far from

Table 9.12 Ratio of the transaction volume of the effective market orders with immediate transactions to the total transaction volume of all orders, based on the simulation and TWSE data

LC range	(-12%,0%]		(0%,1%]		(1%,2%]		(2%,3%]		(3%,4%]		(4%,10%]	
Ticker	Sim	TWSE	Sim	TWSE	Sim	TWSE	Sim	TWSE	Sim	TWSE	Sim	TWSE
0050	26.99%	45.52%	21.39%	6.15%	2.69%	-	0.08%	-	-	-	-	-
0056	13.53%	40.89%	22.7%	9.22%	5.09%	0.01%	1.11%	0.05%	0.28%	-	0.06%	-
01007T	13.51%	44.7%	17.21%	4.17%	7.72%	0.01%	3.28%	0.04%	1.57%	-	1.26%	-
01008T	9.25%	45.44%	13.14%	2.59%	4.42%	0.02%	3.25%	-	2.09%	-	2.94%	-
2002	44.06%	50.68%	20.85%	4.45%	1.21%	0.21%	-	-	-	-	-	-
2330	44.68%	47.84%	19.61%	4.07%	1.96%	0.43%	0.01%	-	-	-	-	-
2454	45.68%	51.43%	20.48%	7.75%	4.54%	0.14%	0.4%	-	-	-	-	-
2498	40.08%	48.64%	19.1%	5.59%	4.99%	0.05%	1.33%	-	0.21%	-	-	-
2912	29.49%	44.41%	18.31%	2.7%	4.31%	0.18%	2.16%	0.01%	1.03%	-	0.74%	-
3474	31.29%	45.84%	17.09%	7.03%	5.41%	0.51%	2.51%	0.35%	0.96%	-	0.28%	-

each other either. However, they have many more transactions with liquidity costs between $0\% - 1\%$ for the simulation data than that for the TWSE data. For an investor, whose main concern is to avert high liquidity costs, the agent-based model produces liquidity costs that are considered to be similar to those for the TWSE data. When devising liquidation strategies for these two securities, this model can be used to simulate liquidity costs under different strategies to identify the optimal ones.

What, then, has distinguished these two securities from others? We examined the data statistics in Table 9.7 and found that they have a high number of transactions. This indicates that this agent-based system simulates liquidity costs more accurately for securities with a higher trading frequency.

9.7 Discussions

The simulated liquidity costs have a wider spread and higher values than those for the TWSE data. This might be due to the following reasons:

1. The average effective market order size rate (σ) used to run the simulation was overestimated. The order-size assumption maintained in this study can also be challenged. In financial literature, order size has been found to follow a power law distribution, and limit order size distribution is different from market order size distribution. By taking into account this difference, we may improve our current simulation.[4]
2. During the simulation, the time interval between two order events is random and independent, which is different from that observed in the real financial markets. Frequently, orders are clustered together in a certain number of time periods, and not evenly distributed throughout a day. This might have contributed to the higher liquidity costs in the simulation data.
3. During simulation, the order events were generated randomly, based on the model parameters, without consulting the order book. This is different from the reality, where an investor normally checks the order book of the opposite side to make sure a profitable matching is possible before submitting an order. In other words, although the order distribution in the simulation system is the same as that for the TWSE (we used the TWSE data to estimate the probability of order submissions), the *sequence* of the order submissions in the simulation system is not optimized as is that devised by human traders.

[4] This direction is pointed out by one of the referees, and is left for future study.

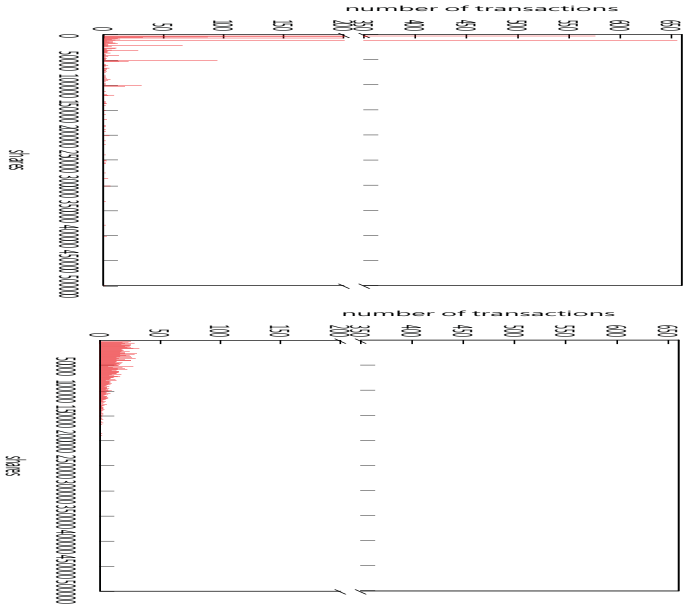


Fig. 9.2 The transaction volume (share) vs. the number of transactions in the TWSE (top) & Simulation (bottom) data

Consequently, the simulated liquidity costs are likely to be higher than those for the TWSE data.

4. In the simulation system, the price of an effective market order is set to be the highest possible bid (buy order) or the lowest possible ask (sell order) allowed by the TWSE to guarantee an immediate transaction. This hardly happens in reality. Normally, a trader would seek a price that generates a transaction, without going to the extreme of the highest possible bid/lowest possible ask. As a result, the simulated liquidity costs are likely to be higher than those for the TWSE data.

To address the first issue, we plan to use two order size parameters, one for the effective market orders rate and one for the effective limit orders, to improve our system. The second issue has been investigated by Engle and Russell [3]. In particular, they devised an Autoregressive Conditional Duration (ACD) model to more realistically simulate the order arrival time, price and volume in a stock market. We are currently integrating this model to our system. This work will be presented in a different publication.

The analysis of items 3 & 4 suggests that traders who employ intelligence (e.g., incorporating order book information) to make trading decisions in a real stock market produced transactions with lower amounts of liquidity costs than that produced by the zero-intelligent agents in our artificial stock market. To simulate the real market behavior, in terms of the liquidity costs, we need to install intelligence (e.g., learning ability) in the artificial agents in our system. We will explore this avenue of research in future work.

One intelligent behavior demonstrated by the TWSE traders is a more profitable liquidation strategy. As shown in Table 9.12, among the daily total trading volume of the Taiwan Top50 Tracker Fund (0050.T), 51.15% are trading volume from the effective market orders with immediate transactions: 26.99% of which paid negative liquidity cost; 21.39% of which

paid liquidity cost between 0 and 1%; 2.69% of which paid liquidity cost of 1-2%; and 0.08% of which paid liquidity cost of 2-3%.

By contrast, the TWSE data show that 51.67% of the daily total trading volume of this security are trading volume from the effective market orders with immediate transactions: 45.52% of which paid negative liquidity cost; and 6.15% of which paid liquidity cost between 0 and 1%. In other words, given the task of liquidating a large block of securities (around 50% of the daily trading volume in this case), the TWSE traders accomplished the task by paying a lower amount of liquidity cost than the cost paid by the zero-intelligence artificial traders. What strategy has delivered such savings?

We analyzed the TWSE 0050.TW transactions data from effective market orders with immediate transactions on March 20, 2008. Figure 9.2 (top) shows that there are many more small-volume transactions than larger-volume ones. In particular, more than 500 transactions are with 5,000 or 10,000 trading shares. This is very different from the simulation data (see the right of Figure 9.2), where the number of small-volume transactions is not dramatically different from that of the large-volume ones (the scale is 20 to 1). This suggests that TWSE traders submitted many smaller-size orders instead of a large-size order to conduct transactions. This strategy has led to a lower amount of liquidity costs. We plan to incorporate this intelligent behavior into the artificial agents in our system.

9.8 Concluding Remarks

The market liquidity of a security plays an important role in financial investment decisions and in the liquidation strategies of the security. As an alternative to the Algorithmic Trading, this study has developed an agent-based model to examine liquidity costs of stocks and securities traded in the Taiwan Stock Market.

For the 10 TWSE stocks and securities that we studied, the model-simulated liquidity costs are higher than those for the TWSE data. We identified four possible factors that contribute to this result:

- The overestimated effective market order size rate.
- The random market order arrival time designed in the DFGIS model.
- The zero-intelligence of the artificial agents in our model.
- The price of the effective market order.

We can continue improving the model by addressing the above-mentioned issues. A model that behaves in a similar way to the TWSE in terms of the liquidity costs can be used to study liquidity costs and to devise liquidation strategies for stocks and securities traded on the TWSE.

Acknowledgements. The authors are grateful to two anonymous referees for their very helpful suggestions. The research support in the form of Taiwan NSC grant no. NSC 98-2410-H-004-045-MY3 is gratefully acknowledged.

References

1. Almgren, R., Chriss, N.: Optimal execution of portfolio transactions. Journal of Risk 3(22), 5–39 (2000)
2. Daniels, M.G., Farmer, J.D., Gillemot, L., Iori, G., Smith, E.: Quantitative model of price diffusion and market friction based on trading as a mechanistic random process. Physical Review Letters 90(10), 108102-1–108102-4 (2003)
3. Engle, R.F., Russell, J.R.: Autoregressive conditional duration: a new model for irregularly spaced transaction data. Econometrica 66(5), 1127–1162 (1998)
4. Farmer, J.D., Patelli, P., Zovko, I.I.: Supplementary material for The Predictive Power of Zero Intelligence in Financial Markets. Proceedings of National Academy of Sciences (USA) 102(6), 2254–2259 (2005)
5. Guo, T.X.: An Agent-Based Simulation of Double-auction Markets. Master's thesis of Graduate Department of Computer Science. University of Toronto (2005)
6. Kalin, D., Zagst, R.: Portfolio optimization under liquidity costs. Working Paper, 04-01, Risklab Germany GmbH, Germany (2004)
7. Weisstein, E.W.: Half-normal Distribution (2005), `http://mathworld.wolfram.com/Half-NormalDistribution.html`

10

Market Microstructure: A Self-Organizing Map Approach to Investigate Behavior Dynamics under an Evolutionary Environment

Michael Kampouridis[1], Shu-Heng Chen[2], and Edward Tsang[1]

[1] School of Computer Science and Electronic Engineering, University of Essex, UK
{mkampo,edward}@essex.ac.uk
[2] AI-Econ Center, Department of Economics, National Cheng Chi University, Taipei, Taiwan
chen.shuheng@gmail.com

Summary. This chapter presents a market microstructure model, which investigates the behavior dynamics in financial markets. We are especially interested in examining whether the markets' behavior is non-stationary, because this implies that strategies from the past cannot be applied to future time periods, unless they have co-evolved with the markets. In order to test this, we employ Genetic Programming, which acts as an inference engine for trading rules, and Self-Organizing Maps, which is used for clustering the above rules into types of trading strategies. The results on four empirical financial markets show that their behavior constantly changes; thus, agents' trading strategies need to continuously adapt to the changes taking place in the market, in order to remain effective.

10.1 Introduction

Market microstructure [36, 37, 38] can be defined as "the study of the process and outcomes of exchanging assets under a specific set of rules. While much of economics abstracts from the mechanics of trading, microstructure theory focuses on how specific trading mechanisms affect the price formation process" [36]. There are several types of models in the agent-based financial markets literature that have focused on market microstructure aspects. One way of categorizing these models is to divide them into the *N*-type models and the autonomous agent models [9]. The former type of model focuses on the mesoscopic level of markets, by allowing agents to choose among different types of trading strategies. A typical example is the fundamentalist-chartist model. Agents in this model are presented with these two strategy types and at any given time they have to choose between these two. A typical area of investigation of these models is fraction dynamics, i.e., how the fractions of the different strategy types change over time. However, what is not presented in most of these models are novelty-discovering agents. For instance, in the fundamentalist-chartists example, agents

A. Brabazon et al. (Eds.): Natural Computing in Comput. Finance, SCI 380, pp. 181–197.
springerlink.com

can only choose between these two types; they cannot create new strategies that do not fall into either of these types. On the other hand, the autonomous agent models overcome this problem by focusing on the microscopic level of the markets. By using tools such as Genetic Programming [26], these models allow the creation and evolution of novel agents, which are not constrained by pre-specified strategy types. However, this kind of models tends to focus on price dynamics, rather than fraction dynamics [9].

In a previous work [10], we combined properties from the *N*-type and autonomous models into a novel model. We first used Genetic Programming (GP) as a rule inference engine, which created and evolved autonomous agents; we then used Self-Organizing Maps (SOM) [25] as a clustering machine, and thus re-created the mesoscopic level that the *N*-type models represent, where agents were categorized into different strategy types. We then investigated the short- and long-term dynamics of the fractions of strategies that existed in a financial market. Nevertheless, that study rested upon an important assumption, i.e., the maps derived from each time period were comparable with each other. This comparability assumption itself required that the types (clusters), as well as their operational specification, would not change over time. If this were not the case, the subsequent study would be questioned. This was mainly due to one technical step in our analysis called translation. The purpose of translation was to place the behavior of agents observed in one period into a different period and to recluster it for the further cross-period comparison. We could not meaningfully have done this without something like topological equivalence, which could not be sustained without the constancy of the types.

However, this assumption can be considered as strict and unrealistic. Strategy types do not necessarily remain the same over time. For instance, if a chartist strategy type exists in time t, it is not certain it will also exist in $t+1$. If market conditions change dramatically, the agents might consider other strategy types as more effective and choose them. The chartist strategy would then stop existing.

In this chapter, we relax the above assumption, since our current work does not require cross-period comparisons. Our model thus becomes more realistic. In addition, *we shift our focus from fraction dynamics to behavior dynamics*: we examine the plausibility of an observation made under artificial markets [1], which suggests that the nature of financial markets constantly changes, or in other words that the markets' behavior is non-stationary [11, 40]. This implies that trading strategies need to constantly co-evolve with the markets; if they do not, they become obsolete or *dinosaurs* [1]. We hence test if this observation holds in the 'real' world, under four empirical financial markets. This will offer important insights regarding the behavior dynamics of the markets.

The rest of this chapter is organized as follows: Section 10.2 presents a brief overview of the different types of agent-based financial models that exist in the literature, and also discusses their limitations. In order to address these limitations, we created an agent-based financial model [10], which is presented in Section 10.3. More specifically, Section 10.3 presents the two techniques that our agent-based model uses, namely Genetic Programming and Self-Organizing Maps. In addition, Sect. 10.3 gives some background information on these two techniques, and also discusses the details of the algorithms we have used. Section 10.4 then presents the experimental designs, Section 10.5 reviews the testing methodology, and Sect. 10.6 presents the results of our experiments. Finally, Section 10.7 concludes this chapter and discusses future work.

10.2 Agent-Based Financial Models

Agent-based financial models are models of financial markets, where artificial agents can trade with each other. These models simulate the simultaneous operations and interactions of the different agents that exist in the market, with the goal of re-creating and predicting the appearance of complex phenomena. Building such models can give valuable information about different aspects of market dynamics, such as behavior dynamics [7]. As we have already mentioned, [9] divides these models into two basic designs: the N-type design, and the autonomous agent design. The rest of this section presents these two designs.

10.2.1 N-Type Designs

Two- and Three-Type Designs

In this type of designs, agents have beliefs regarding the price of a stock in the next period. In the two-type design, there are two types of agent beliefs. Consequently, there are two types of *fixed and pre-specified* trading strategies. Each agent *can only choose between these two types*. These two types are usually fundamental and technical traders.[1]

The three-type design is an extension of the two-type one, where there are three types of agents. One way to implement this design is to have two types of chartists, momentum traders and contrarian traders [42]. The former is the kind of agent we described above as 'chartists'. The latter, the contrarian traders, extrapolate past movements of the price into the future, the opposite way that the trend goes. This happens because contrarians believe that the price trend will finish soon and will start to reverse.

Finally, we should mention that several extensions of the above designs exist, by enriching their behavioral rules. For instance, a typical way to do this is by adding a memory factor to these rules. Other extensions can be to add an adaptive behavior, where the agents can learn from their previous experiences. Such examples can be found in [5], where Brock and Hommes use 2-, 3-, and 4-type models. Other adaptive behavior examples include Kirman's ANT Model [23, 24] and Lux's Interactive Agent Hypothesis Model [32, 33, 34].

Many-Type Designs

So far we have seen designs with few 'fixed' types, namely two and three. However, in the literature we can find other N-type designs, where $N > 3$.

Adaptive Belief Systems
A very good example of many-type design is the Adaptive Belief System (ABS) of Brock and Hommes [4, 5]. This system can be considered as an extension of the two- and three-type designs we have seen. The number of strategies takes values from 1 to N and these are known and fixed, like before. This means that agents can choose from a finite and fixed number of beliefs.

Large Type Limit and Continuous Belief Systems
Other many-type designs include the Large Type Limit (LTL) [6] and the the Continuous Belief System (CBS) [12]. In these systems, the number N of strategies is not finite, but infinite, i.e., $N \to \infty$. Both of these systems are based on an idea called *distribution of beliefs*, where there is a belief space from which the observed beliefs are sampled.

[1] Other equivalent names for technical traders are chartists, trend-followers and noisy traders.

10.2.2 Autonomous Agent Designs

So far, we have talked about *N*-type designs, where the trading strategies are pre-specified and fixed by the model designer. Thus, the agents are restricted in using these specific strategies and cannot come up with any new ones. Although the *N*-type design had been characterized as a major class of agent-based financial models, it has also been agreed that it severely restricts the degree of autonomy available for financial agents [9].

This issue of autonomy was addressed by the autonomous agent designs, where we can have artificial agents who are autonomous and thus have the ability to discover new strategies, which have never been used before. An example of this is the well-known Santa Fe Institute (SFI) model [2, 39], where a Genetic Algorith (GA) [17] was used. Thus, a fixed number of strategies does not exist; on the contrary, each artificial agent can have a different trading strategy which is "customized" by a GA. The SFI model is of course not the only application of GA in artificial stock markets. Another example is AGEDASI TOF[2] [18, 19]. If the reader is interested in these topics, a very good literature review can be found in [8].[3]

10.2.3 Limitations of the Agent-Based Financial Models

In the previous sections, we described the two main agent-based financial designs: the *N*-type and the SFI-like (autonomous agents design) ones. The former design consists of *N* pre-specified strategy types, and the agents have to choose among these *N* types. An advantage of this design is that it allows us to observe the changes in the market fraction[4] dynamics of the above strategy types. However, as we saw, a disadvantage of this type of model is that the agents are restricted in choosing from the given *N* strategy types. In addition, another limitation of this type of model is the lack of heterogeneity. Agents that belong in the same trading strategy type follow *exactly the same behavioral rule*. Nevertheless, in the real world, the behavior of each trader is expected to be heterogeneous; even if some traders are following a certain trading strategy type, it does not mean that they behave in exactly the same way.

As we saw, the issue of heterogeneity is addressed by SFI-like models. This type of model allows the creation of autonomous and heterogeneous agents. Nevertheless, even under the autonomous agent model, agents have to choose among a pre-specified number *N* of trading strategy types [9]. To the best of our knowledge, there is no model that uses autonomous agents that are not restricted to predefined, fixed strategy types.

This thus motivated us to create such a model. In order to do this, we used Genetic Programming as a rule inference engine, and Self Organizing Maps as a clustering tool. The next section presents our model in detail.

10.3 Model

In this section, we present our agent-based financial model, which was first introduced in [10]. This model first allows the creation of novel, autonomous and heterogeneous agents by the

[2] It stands for A GEnetic-algorithmic Double Auction SImulation in TOkyo Foreign exchange market

[3] It should also be said that apart from GA, other population-based learning models have been used, such as GP. We refer the readers to [9] for more details.

[4] Market fraction refers to the proportion of the different trading strategy types (e.g., fundamentalists and chartists) that exist in a financial market.

use of GP. The reason for using GP is because the market is considered as an evolutionary process; this is inspired by Andrew Lo's Adaptive Market Hypothesis (AMH) [30, 31], where Lo argued that the principles of evolution (i.e., competition, adaptation, and natural selection) can be applied to financial interactions. Thus, agents can be considered to be organisms that learn and try to survive.

After creating and evolving novel agents, we cluster them into types of trading strategies via a SOM. These types are thus not pre-specified, but depend on the strategies of the agents. In this way, we are able to reconstruct the microscopic level of markets (SFI-like designs), where financial agents are created, and connect it to the mesoscopic level (*N*-type designs), where agents are clustered into *N* strategy types. Figure 10.1 illustrates this process.

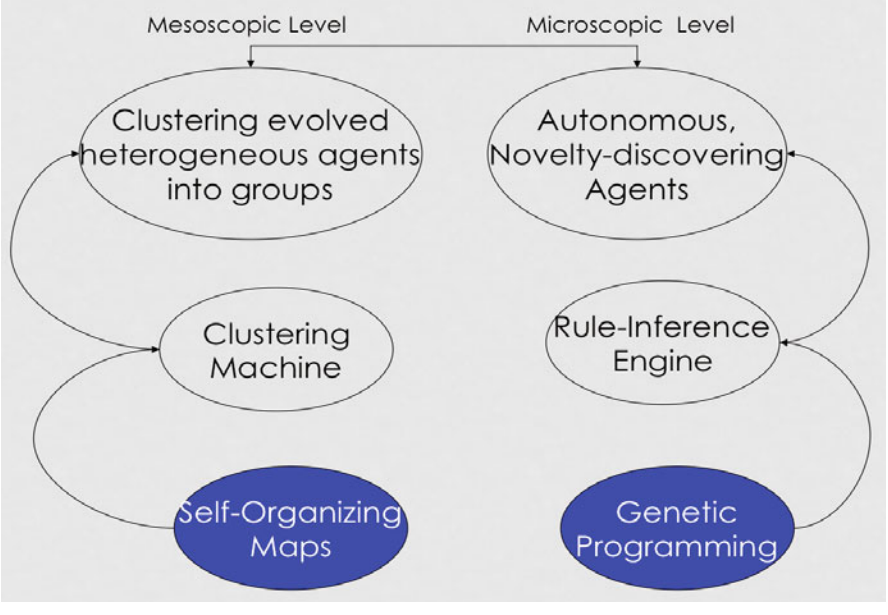


Fig. 10.1 Connecting the microscopic to the mesoscopic level of financial markets

In the rest of Section 10.3 we give some brief background information on GP and SOM, explain how they were used in our agent-based model, and also present the algorithms we used behind these techniques.

10.3.1 Genetic Programming (GP)

Genetic Programming (GP) [3, 26, 27, 28, 29, 41] is an evolutionary technique inspired by natural evolution, where computer programs act as the individuals of a population. The GP process has several steps. To begin with, a random population is created, by using terminals and functions appropriate to the problem domain, where the former are variables and constants of the programs, and the latter are responsible for processing the values of the system (either terminals or other functions' output).

In the next step, each individual is measured in terms of a pre-specified fitness function. The purpose of assigning a fitness to each individual is to measure how well it solves the problem. In the following step, individuals are chosen to produce new offspring programs. A typical way of doing this is by using the fitness-proportionate selection, where an individual's probability of being selected is equal to its normalized fitness value [26]. The individuals chosen from the population are manipulated by genetic operators such as crossover and mutation, in order to produce offspring. The new offspring constitute the new population. Finally, each individual in the new population is again assigned a fitness and the whole process is iterated, until a termination criterion is met (usually a set number of generations). At the end of this procedure (last generation), the program with the highest fitness is considered as the result of that run. Next, we explain how GP was utilized in our model.

GP as a Rule-Inference Engine

First of all, we assume that traders' behavior, including price expectations and trading strategies, is either not observable or not available. Instead, their behavioral rules have to be estimated by the observable market price. Using macro data to estimate micro behavior is not new,[5] as many empirical agent-based models have already performed such estimations [9]. However, such estimations are based on very strict assumptions, as we saw earlier (e.g., having pre-specified trading strategy types is considered to be a strict and unrealistic assumption). Since we no longer keep these assumptions, an alternative must be developed, and in this study we adopt Genetic Programming (GP).

As we have already mentioned, the use of GP is motivated by considering the market as an evolutionary and selective process.[6] In this process, traders with different behavioral rules participate in the markets. Those behavioral rules which help traders gain lucrative profits will attract more traders to imitate, and rules which result in losses will attract fewer traders. An advantage of GP is that it does not rest upon any pre-specified class of behavioral rules, like many other models in the agent-based finance literature [9]. Instead, in GP, a population of behavioral rules is randomly initiated, and the survival-of-the-fittest principle drives the entire population to become fitter and fitter in relation to the environment. In other words, given the non-trivial financial incentive from trading, traders are aggressively searching for the most profitable trading rules. Therefore, the rules that are outperformed will be replaced, and only those very competitive rules will be sustained in this highly competitive search process.[7]

Hence, even though we are not informed of the behavioral rules followed by traders at any specific time horizon, GP can help us infer what are the rules the traders follow, by simulating the evolution of the microstructure of the market. Traders can then be clustered

[5] 'Macro data' is generally a term used to mainly describe two categories of data: aggregated data, and system-level data [13]. The former refers to data that combine information, such as unemployment statistics and demographics. The latter refers to information that cannot be disaggregated to lower level unities; such examples are the prices of a stock. On the other hand, 'micro behavior' refers to the study of the behavior of components of a national economy, such as individual firms, households and traders.

[6] See [30, 31] for his eloquent presentation of the *Adaptive Market Hypothesis*.

[7] It does not mean that all types of traders surviving must be smart and sophisticated. They can be dumb, naive, randomly behaved or zero-intelligent. Obviously, the notion of rationality or bounded rationality applying here is *ecological* [16, 43].

based on realistic, and possibly complex behavioral rules.[8] The GP algorithm used to infer the rules is presented in detail in the next section.

GP Algorithm

Our GP is inspired by a financial forecasting tool, EDDIE [20], which applies genetic programming to evolve a population of market-timing[9] strategies, which guide investors on when to buy or hold. These market timing strategies are formulated as decision trees, which, when combined with the use of GP, are referred to as *Genetic Decision Trees* (GDTs). Our GP uses indicators commonly used in technical analysis: Moving Average (MA), Trader Break Out (TBR), Filter (FLR), Volatility (Vol), Momentum (Mom), and Momentum Moving Average (MomMA).[10] Each indicator has two different periods, a short- and a long-term one (12 and 50 days). Figure 10.2 presents a sample GDT generated by the GP. As we can observe, this tree suggests that the trader should buy if the 12 days Moving Average is less than 6.4. If, however, this is not the case, the tree examines the 50 days Momentum; if it is greater than 5.57, the then GDT recommends not-to-buy. If, finally, the 50 days Momentum is less than or equal to 5.57, then the GDT recommends to buy.

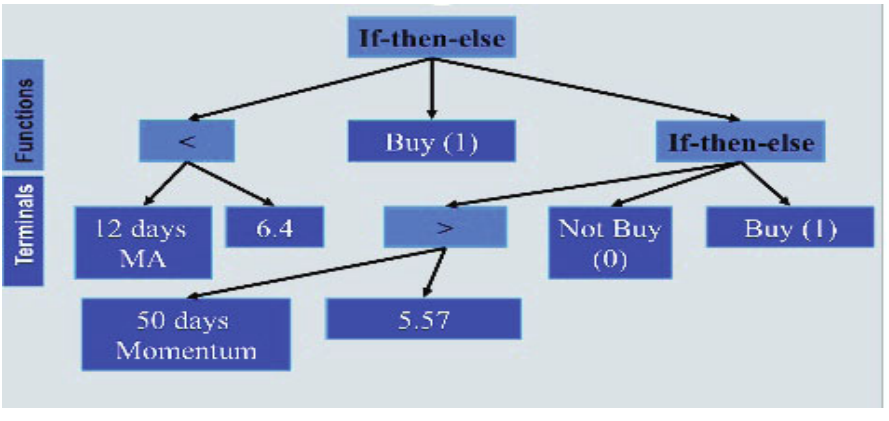


Fig. 10.2 Sample GDT generated by the GP

Depending on the classification of the predictions, there are four cases: True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN).

[8] Duffy and Engle-Warnick [15] provides the first illustration of using genetic programming to infer the behavioral rules of human agents in the context of ultimatum game experiments.

[9] 'Market timing' refers to the strategy of making buy or sell decisions of stocks, by attempting to predict future price movements.

[10] We use these indicators because they have been proved to be quite useful in previous works like [20]. However, the purpose of this work is not to provide a list of the ultimate technical indicators.

We then use the following 3 metrics, presented in Equations (10.1)-(10.3):

Rate of Correctness

$$RC = \frac{TP+TN}{TP+TN+FP+FN} \tag{10.1}$$

Rate of Missing Chances

$$RMC = \frac{FN}{FN+TP} \tag{10.2}$$

Rate of Failure

$$RF = \frac{FP}{FP+TP} \tag{10.3}$$

The above metrics combined give the following fitness function:

$$ff = w_1 * RC - w_2 * RMC - w_3 * RF \tag{10.4}$$

where w_1, w_2 and w_3 are the weights for RC, RMC and RF, respectively, and are given in order to reflect the preferences of investors. For instance, a conservative investor would want to avoid failure; thus a higher weight for RF should be used. For our experiments, we chose to include GDTs that mainly focus on correctness and reduced failure. Thus these weights have been set to 1, $\frac{1}{6}$ and $\frac{1}{2}$, respectively.

Given a set of historical data and the fitness function, GP is then applied to evolve the market-timing strategies in a standard way. After evolving a number of generations, what survives at the last generation is, presumably, a population of financial agents whose market-timing strategies are financially rather successful. We then use SOM to cluster these strategies into types of trading strategies.

10.3.2 Self Organizing Maps (SOM)

Self-Organizing Maps (SOM) [25] are a type of artificial neural networks which takes as input data with high dimensionality,[11] and returns a low-dimensional representation of these data, along with their topological representation. This representation is called a map. A self-organizing map consists of components called neurons. Associated with each neuron is a weight vector, which has the same dimensions as the input data. During the SOM procedure, the weight vector of each neuron is dynamically adjusted via a competitive learning process. Eventually, each weight vector becomes the center (a.k.a. centroid) of a cluster of input vectors. Thus, at the end of the SOM procedure, all input vectors have been assigned to different clusters of a map. The next section presents how SOM was applied to our model.

SOM as a Clustering Machine

Once a population of rules is inferred from GP, it is desirable to cluster them based on a chosen similarity criterion. As we have already discussed at the beginning of Section 10.3, this allows us to cluster heterogeneous agents into different types of trading strategies, which are neither fixed, nor pre-specified.

The similarity criterion which we choose is based on the *observed trading behavior*.[12] Based on this criterion, two rules are similar if they are *observationally equivalent* or

[11] In this work the input data is the market-timing vectors of the GDTs.

[12] Other similarity criteria could be used such as risk averseness. However, in this study we wanted to focus on the behavioral aspects of the rules.

similar, or, alternatively put, they are similar if they generate the same or similar market timing behavior.[13]

Given the criterion above, the behavior of each trading rule can be represented by its series of market timing decisions over the entire trading horizon, for example, 6 months. Therefore, when we denote the decision "buy" by "1" and "not-buy" by "0", then the behavior of each rule (GDT) is a binary vector. The dimensionality of these vectors is then determined by the length of the trading horizon. For example, if the trading horizon is 125 days long, then the dimension of the market timing vector is 125. Thus, each GDT can be represented by a vector which contains a series of 1s and 0s, denoting the tree's recommendations to buy or not-buy on each day. Once each trading rule is concretized into its market timing vector, we can then easily cluster these rules by applying Kohonen's Self-Organizing Maps to the associated clusters.

The main advantage of SOMs over other clustering techniques such as K-means [35] is that the former can present the result in a visualizable manner, so that we can not only identify these types of traders, but also locate their 2-dimensional position on a map, i.e., a distribution of traders over a map. This provides us with a rather convenient grasp of the dynamics of the microstructure directly as if we were watching the population density on a map over time.

SOM Algorithm

For our experiments, we use MathWorks' Neural Network toolbox,[14] which is built in the MATLAB environment. We refer the reader to the relevant documentation for details on the algorithm. Figure 10.3 presents a 3×3 SOM which has been produced by this toolbox. Here, 500 artificial traders are grouped into nine clusters (types of trading strategies). In a sense, this could be perceived as a snapshot of a nine-type agent-based financial market dynamics. Traders of the same type indicate that their market timing behavior is very similar. The market fraction or the size of each cluster can be seen from the number of traders belonging to that cluster. Thus, we can observe that the largest cluster has a market share of 71.2% (356/500), whereas the smallest one has a market share of 0.2% (1/500).

10.4 Experimental Designs

The experiments are conducted for a period of 17 years (1991-2007) and the data are taken from the daily closing prices of 4 international market indices: NYSE (USA), S&P 500 (USA), STI (Singapore), and TAIEX (Taiwan). For statistical purposes, we repeat our experiments 10 times.

Each year is split into 2 halves (January-June, July-December), so in total, out of the 17 years, we have 34 periods.[15] The first semester of a year is denoted with an 'a' at the end (e.g., 1991a), and the second semester of a year is denoted with a 'b' (e.g., 1991b). The GP

[13] One might question the above similarity criterion, since very different rules might be able to produce the same signals. This does not pose a problem in this work, since we are interested in the behavior of the market (and thus the rules' behavior). We are not interested in the semantics aspect of the rules.

[14] http://www.mathworks.com/access/helpdesk/help/toolbox/nnet/self_or4.html

[15] At this point the length of the period is chosen arbitrarily as 6 months. We leave it to future research to examine if and how this time horizon can affect our results.

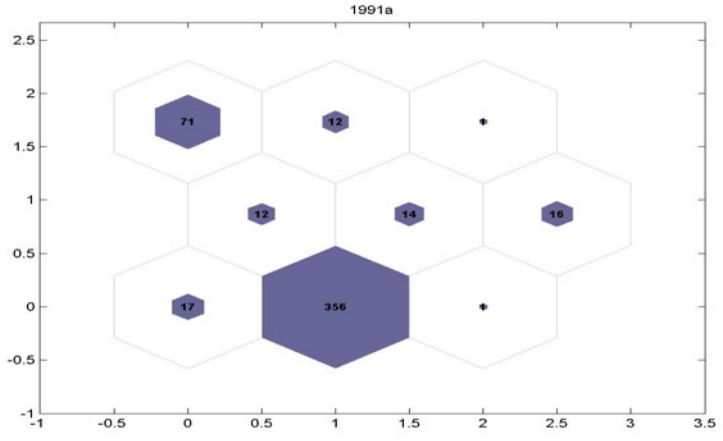


Fig. 10.3 Example of a 3×3 Self-Organizing Map

system is therefore executed 34 times, i.e., one time per period. Table 10.1 presents the GP parameters for our experiments. The GP parameters for our experiments are the ones used by Koza [26]. Only the tournament size has been lowered, because we were observing premature convergence. Other than that, the results seem to be insensitive to these parameters. As we can observe, we have a population of 500 trading strategies. Thus, since there are 34 periods, we end up with 34 different populations of 500 strategies.

Table 10.1 GP Parameters

GP Parameters	
Max Initial Depth	6
Max Depth	17
Generations	50
Population size	500
Tournament size	2
Reproduction probability	0.1
Crossover probability	0.9
Mutation probability	0.01

After generating and evolving strategies for each one of the 34 periods, we then use SOM to cluster these strategies into types. We do this for every one of the 34 periods. Thus, we end up with 34 different SOMs, one per semester, which represent the market in different time periods over the 17-year horizon.

Table 10.2 presents the SOM parameters for our experiments. The parameters are the default ones provided in MATLAB's MathWorks Neural Network Toolbox.

Table 10.2 Default SOM parameters of the MathWorks SOM Toolbox

SOM Parameters	
Algorithm	Batch
Distance	Euclidean
Neighborhood Radius	$\sigma = 3$
Topology	Hexagonal
Steps	100

Finally, we define as 'base period', the period during which GP creates and evolves GDTs. We also define 'future period(s)', as the period(s) which follow(s) the base period (in chronological order).

10.5 Testing Methodology

As we mentioned at the beginning of this chapter, we are interested in investigating Arthur's observation about the constantly changing behavior of financial markets [1]. In order to investigate whether the behavior of markets is non-stationary, we recluster the GDTs of each base period, to all future periods' clusters.[16] By applying the same GDTs (strategies) to clusters of future periods, we can observe how well these strategies fit in the new environments (clusters). The logic behind this is the following: when we first evolved and clustered the GDTs (base period), these GDTs were placed in clusters that represented their respective strategies. For instance, if there was a strategy type (cluster) that represented 'chartists', then all GDTs which followed a chartist strategy were placed in that cluster. When we then take the GDTs from a base period and recluster them to strategy types of future periods, it is not guaranteed that there will again be a cluster that represents chartists. If the market constantly changes, there is a possibility that this type of strategies does not exist any more in the future periods. Thus, the GDTs find themselves *unadapted* to the new environment (clusters) and have to choose another cluster, which represents them as closely as possible. This cluster will be the one that has the centroid with the smallest Euclidean distance[17] from the market-timing vectors of these GDTs. Of course, since now the SOM of the future period is formed by different clusters, the GDTs might not fit in as well as they did in the base period. In order to measure this 'unfitting', we use a *dissatisfaction rate*, i.e., how dissatisfied these GDTs will be when placed into a future period's cluster that does not represent their strategy. *If the market is non-stationary, the GDTs' dissatisfaction rate will be high*, as a result of the changes that took place in the market. The dissatisfaction rate is defined as the Euclidean distance of a GDT's market-timing vector to the centroid of the cluster in which it is placed, after the reclustering procedure. Under a non-stationary market behavior, the following statement should hold:

The average dissatisfaction rate of the population of GDTs from future periods should not return to the range of dissatisfaction of the base period.

Hence, we will test the above statement against the 4 financial markets.

[16] The process of reclustering is explained later in this section.

[17] One may wonder if the choice of the Euclidean distance as a distance metric, when the vectors of the GDTs are binary, is an appropriate one. However, this does not pose a problem, because the vectors of the clusters' centroids are real valued.

Let us now explain the process of reclustering. We start with 1991a as the base period. Each evolved GDT is moved to the next period, 1991b, and reclustered into one of the clusters of that period. In order to 'decide' which cluster to choose, the GDT compares the Euclidean distance of its market timing vector to the centroid of each cluster; it is then placed into the cluster with the smallest Euclidean distance. The same procedure follows for all GDTs of the population. At the end, the population of evolved GDTs from the base period of 1991a will have been reclustered into the clusters of period 1991b. The same procedure is followed in all future periods. This means that the GDTs from 1991a are also reclustered into 1992a, 1992b, ..., 2007b. Finally, the same process is done for all other base periods (i.e., 1991b, 1992a, ..., 2007a).

Once the process of reclustering is complete, we calculate the dissatisfaction rate of each GDT in the population. Next, we calculate the population's average dissatisfaction rate. We do the same for all 34 periods. Given a base period, the population average dissatisfaction of all periods is normalized by dividing those population average dissatisfaction rates by the population average dissatisfaction rate in the base period. For instance, if the population dissatisfaction rates for periods 1991a, 1991b, 1992a, ..., 2007b are 0.8, 0.85, 0.9, ..., 0.85, respectively, the the normalized population dissatisfaction for the base period 1991a would be $\frac{0.8}{0.8}$, $\frac{0.85}{0.8}$, $\frac{0.9}{0.8}$,..., $\frac{0.85}{0.8}$. Hence, each base period has its normalized average dissatisfaction rate equal to 1. In order to prove that the market is non-stationary, we need to show that the normalized average dissatisfaction rate of the GDTs increases in the future periods, and never returns to its initial value of 1, which was during the base period. If, on the other hand, this rate reaches 1 or below, *it is an indication of a cyclic market behavior*, since the GDTs have found the same conditions with the base period, and as a result feel as 'satisfied' as before.

Finally, we define as *dinosaurs* the population of GDTs that has been reclustered from a base period to future periods. The reason of calling them in this way is because these GDTs have not adapted to the new market environment (clusters of the SOMs from future periods) and are thus ineffective. If these GDTs' normalized average dissatisfaction rate drops to *less than or equal to 1*, we call them *returning dinosaurs*, because they have become effective again.[18]

10.6 Results

As explained, returning dinosaurs denote a cyclic market behavior. To examine whether dinosaurs return, we iterate through each base period and calculate the minimum normalized average dissatisfaction rate for each future period. This gives us an indication of how many returning dinosaurs, if any, exist. If, for instance, 1991a is the base period, then there is a series of 33 population dissatisfaction values for its future periods. We obtain the minimum value among these 33 values, in order to check how close to 1 the normalized dissatisfaction rate of this future period is. This process is then repeated for 1991b as the base period and its 32 future periods, and so on, until base period 2007a. We thus end up with a 1×33 vector, which presents the minimum dissatisfaction per base period and thus shows whether any returning dinosaurs exist. In addition, we are interested in investigating whether different number of

[18] In a previous work [21], where we investigated the markets' behavior dynamics by only using GP but not SOM, we did not use this 'strict' definition of returning dinosaurs. This led us to conclude that returning dinosaurs existed. However, if we had also used the current study's definition, the results from [21] would not have dramatically differed from those of this chapter.

clusters (strategy types) can affect the test results. We thus run tests under 2 to 9 clusters, for the following SOM dimensions: 2×1, 3×1, 2×2, 5×1, 3×2, 7×1, 4×2, and 3×3. Figure 10.4 presents the graphs of the minimum dissatisfaction vectors for the 4 international markets. Each line represents the results of a different SOM dimension. The horizontal line indicates a dissatisfaction of 1, and is given as a reference. Results are the average of 10 runs.

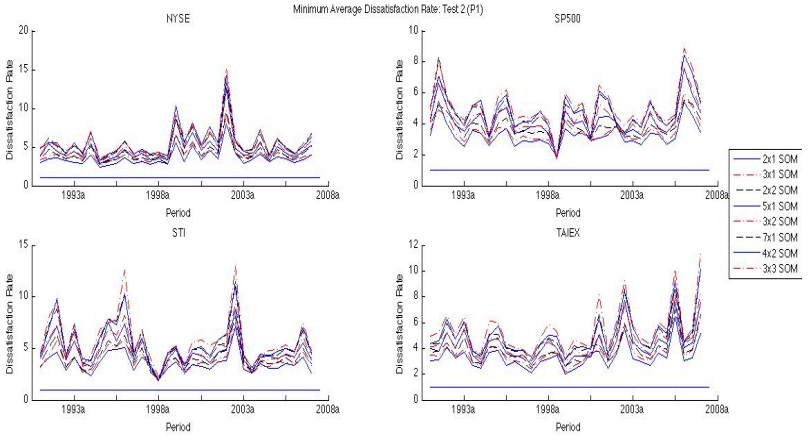


Fig. 10.4 Minimum normalized dissatisfaction rate of the population of GDTs per base period, for all SOM dimensions, for all datasets. Each subfigure represents a single dataset. From left to right, top to bottom: NYSE, S&P 500, STI, TAIEX

What we can see from Figure 10.4 is that there are no base periods with a minimum normalized dissatisfaction rate below 1. This observation holds under all SOM dimensions, and under all of the financial markets tested in this study. In fact, the closest to 1 that the dissatisfaction rate gets is around 2, for S&P 500 (1999a), STI (1998a), and TAIEX (1999b). But even these periods could be treated as exceptional cases, because it is obvious that the average dissatisfaction rate is much higher than 2.

To make the above argument even clearer, we present the average dissatisfaction rates over the 10 runs in Table 10.3. Each row presents the average dissatisfaction rate for a different market, and each column presents the rate under a different SOM dimension. As we can observe, these values range from 4.44 (NYSE-2×1 SOM) to 8.88 (STI-3×3 SOM). In addition, the average dissatisfaction rate ranges from 4.75 (2 clusters) to 7.85 (9 clusters). It is thus obvious that *on average, no dinosaurs return*, because the average dissatisfaction rate is much higher than 1. But even if we want to take into account the outliers (minimum dissatisfaction rate), not many things change. Table 10.4 presents the average, over the 10 runs, of the minimum dissatisfaction rates. As we can see, these rates range from 3.28 (S&P 500-2×1 SOM) to 5.79 (STI-3×3 SOM). Furthermore, the average of the minimum dissatisfaction rate per SOM dimension is 3.41 for the 2×1 SOM, and it gradually increases, as the number of clusters increases, reaching 5.46 for the 3×3 SOM. Hence, the minimum dissatisfaction rate is on average quite far away from 1, which as we mentioned is the threshold for a returning dinosaur.

Table 10.3 Average of Average Dissatisfaction Rate per Cluster per Dataset

	2×1	3×1	2×2	5×1	3×2	7×1	4×2	3×3
NYSE	4.44	4.90	5.22	5.92	6.17	6.88	6.96	7.27
S&P500	4.36	4.61	4.93	5.53	5.79	6.45	6.56	6.86
STI	5.17	5.65	6.11	6.98	7.19	8.30	8.33	8.88
TAIEX	5.04	5.48	5.74	6.54	6.96	7.76	7.78	8.40
Mean	4.75	5.16	5.50	6.24	6.53	7.35	7.41	7.85

Table 10.4 Average of Minimum Dissatisfaction Rate per Cluster per Dataset

	2×1	3×1	2×2	5×1	3×2	7×1	4×2	3×3
NYSE	3.52	3.94	4.14	4.69	4.98	5.33	5.53	5.56
S&P500	3.28	3.48	3.77	4.18	4.31	4.78	4.80	5.07
STI	3.56	3.83	4.09	4.61	4.79	5.34	5.41	5.79
TAIEX	3.29	3.64	3.83	4.25	4.44	4.88	5.00	5.43
Mean	3.41	3.72	3.96	4.43	4.63	5.08	5.19	5.46

An observation we can make from the above results is that the trading strategies never reach a dissatisfaction rate of 1. Thus, *returning dinosaurs do not exist*. Market conditions constantly change and the strategies cannot be as "satisfied" as they were in the base period. However, there can be a few exceptions, where the dissatisfaction rate goes quite low, e.g. around 2. Although we cannot say that dinosaurs have returned, it is obvious that the market in those years has similarities to the base period. In these cases, we can say that *dinosaurs have returned as lizards*. Nevertheless, strategies that have not co-evolved with the market, cannot reach performance levels as the ones they once had in their base period (i.e., no returning dinosaurs). Market conditions change continuously.

The above observations are very important and allow us to argue that the behavior of the 4 markets tested in this study constantly change. Trading strategies need to adapt to these changes, in order to remain effective. If they do not, they find the new environment (clusters) very different from the one in the base period and thus are very "dissatisfied". These strategies thus become obsolete or dinosaurs.

One final observation we can make is that the number of clusters does not affect the test's results. The dissatisfaction rate of each market follows always the same pattern, regardless the number of clusters. No returning dinosaurs are observed, under any number of the trading strategy types tested.

10.7 Conclusion

To conclude, this chapter presented a significant extension to a previous market microstructure model [10], and also discussed preliminary results on the behavior dynamics of financial markets.

In [10], we used Genetic Programming (GP) as a rule-inference engine to find out the behavioral rules of agents, and Self-Organizing Maps (SOM) to cluster these agents. However, because of an important assumption in that work, we had to require that SOM clusters, as well as their operational specification, would remain the same over time. In this chapter, we relaxed that assumption. This offered more realism to our model and allowed us to investigate market behavior dynamics.

Our experimental work was inspired by an an observation made under artificial agent-based financial markets [1]. This observation says that the nature and constituents of agents, and thus their strategies, constantly change; if these strategies do not continuously adapt to the changes in their environments, then they become obsolete (dinosaurs). In order to test the plausibility of this observation, we ran tests on 4 international financial markets.

The results showed that on average, these datasets did not demonstrate the existence of returning dinosaurs, and thus *verified the existence of the non-stationary property in financial markets' behavior*. The implications of this are very important. Strategies from the past cannot be successfully re-applied to future periods, unless they have co-evolved with the market. If they have not, they become obsolete, because the market conditions change continuously. They can occasionally return as lizards, meaning that these strategies can sometimes demonstrate relatively good performance, but they cannot become again as successful, as they once were.

Future work will focus on exploring whether the above observations can hold under even more financial markets. In addition, we aim to show that our experimental results are independent from the GP and SOM algorithms we have used in this chapter. We have already done some work towards this direction, where we ran experiments under 2 different GP algorithms and found that results were independent to the choice of the algorithm [22]. Moreover, we are also interested in demonstrating that our results would be the same if we were using different rule inference engines and clustering machines. To show this, we aim to run experiments where we use a Genetic Algorithm instead of Genetic Programming and other clustering techniques, such as standard hierarchical clustering [44] and growing hierarchical SOMs [14], instead of the classical SOM approach.

Acknowledgments. The version has been revised in light of two anonymous referees' very helpful reviews, for which the authors are most grateful. The EPSRC grant (No. EP/P563361/01) and the NSC grant 98-2410-H-004- 045-MY3 are also gratefully acknowledged.

References

1. Arthur, B.: On learning and adaptation in the economy, working paper 92-07-038. Santa Fe Institute (1992)
2. Arthur, W., Holland, J., LeBaron, B., Palmer, R., Tayler, P.: Asset pricing under endogenous expectations in an artificial stock market. In: Arthur, B., Durlauf, S., Lane, D.E. (eds.) The Economy as an Evolving Complex System II, pp. 15–44. Addison-Wesley, Reading (1997)
3. Banzhaf, W., Nordina, P., Keller, R., Francone, F.: Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and its Applications. Morgan Kaufmann, Heidelberg (1998)
4. Brock, W., Hommes, C.: A rational route to randomness. Econometrica 65, 1059–1095 (1997)

5. Brock, W., Hommes, C.: Heterogeneous beliefs and routes to chaos in a simple asset pricing model. Journal of Economic Dynamics and Control 22, 1235–1274 (1998)
6. Brock, W., Hommes, C., Wagener, F.: Evolutionary dynamics in markets with many trader types. Journal of Mathematical Economics 41, 7–42 (2005)
7. Chan, N., LeBaron, B., Lo, A., Poggio, T.: Agent-based models of financial markets: A comparison with experimental markets. MIT Artificial Markets Project Paper No.124 (1999)
8. Chen, S.H., Huang, Y.C., Wang, J.F.: Bounded rationality and the elasticity puzzle: An analysis of agent-based computational consumption capital asset pricing models. In: Zambelli, S. (ed.). Routledge, New York (2009)
9. Chen, S.H., Chang, C.L., Du, Y.R.: Agent-based economic models and econometrics. Journal of Knowledge Engineering Review (2010) (forthcoming)
10. Chen, S.H., Kampouridis, M., Tsang, E.: Microstructure dynamics and agent-based financial markets. In: Bosse, T., Geller, A., Jonker, C.M. (eds.) MABS 2010. LNCS(LNAI), vol. 6532, pp. 121–135. Springer, Heidelberg (2011)
11. Dickey, D., Fuller, W.: Distribution of the estimates for autoregressive time series with a unit root. Journal of the American Statistical Association 74, 427–431 (1979)
12. Dicks, C., Van der Weide, R.: Herding asynchronous updating and heterogeneity in memory in a CBS. Journal of Economic Dynamics and Control 29(4), 741–763 (2005)
13. Diez-Roux, A.: A glossary for multilevel analysis. Journal of Epidemiology and Community Health 56, 588–594 (2002)
14. Dittenbach, M., Rauber, A., Merkl, D.: Recent advances with the growing hierarchical self-organizing map. In: Allinson, N., Yin, H., Allinson, L., Slack, J. (eds.) Proceedings of the 3rd Workshop on Self-Organizing Maps. Advances in Self-Organizing Maps, pp. 140–145. Springer, Lincoln (2001)
15. Duffy, J., Engle-Warnick, J.: Using symbolic regression to infer strategies from experimental data, In: pp. 61–82. Springer, Heidelberg (2002); Evolutionary Computation in Economics and Finance
16. Gigerenzer, G., Todd, P.: Fast and Frugal Heuristics: The Adaptive Toolbox. In: Gigerenzer, G., Todd, P. (eds.), pp. 3–34. Oxford University Press, Oxford (1999); The ABC Research Group
17. Holland, J.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975)
18. Izumi, K., Okatsu, T.: An artificial market analysis of exchange rate dynamics. In: Fogel, L.J., Angeline, P.J. (eds.) Evolutionary Programming V, pp. 27–36. MIT Press, Cambridge (1996)
19. Izumi, K., Ueda, K.: Analysis of dealers' processing financial news based on an artificial market approach. Journal of Computational Intelligence in Finance 7, 23–33 (1999)
20. Kampouridis, M., Tsang, E.: EDDIE for investment opportunities forecasting: Extending the search space of the GP. In: Proceedings of the IEEE Conference on Evolutionary Computation, Barcelona, Spain, pp. 2019–2026 (2010)
21. Kampouridis, M., Chen, S.H., Tsang, E.: Testing the dinosaur hypothesis under empirical datasets. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6239, pp. 199–208. Springer, Heidelberg (2010)
22. Kampouridis, M., Chen, S.H., Tsang, E.: Investigating the effect of different GP algorithms on the non-stationary behavior of financial markets. In: Computational Intelligence for Financial Engineering and Economics. IEEE Symposium Series on Computational Intelligence. IEEE Press, Los Alamitos (2011) (forthcoming)
23. Kirman, A.: Epidemics of Opinion and Speculative Bubbles in Financial Markets. In: Taylor, M. (ed.) Money and Financial Markets, pp. 354–368. Macmillan, London (1991)

24. Kirman, A.: Ants, rationality and recruitment. Quarterly Journal of Economics 108(1), 137–156 (1993)
25. Kohonen, T.: Self-organized formation of topologically correct feature maps. Journal of Biological Cybernetics 43, 59–69 (1982)
26. Koza, J.: Genetic Programming: On the programming of computers by means of natural selection. MIT Press, Cambridge (1992)
27. Koza, J.: Genetic Programming II: Automatic Discovery of Reusable Programs. MIT Press, Cambridge (1994)
28. Koza, J., Andre, D., Bennett III, F., Keane, M.: GeneticProgramming 3: Darwinian Invention and Problem Solving. Morgan Kaufmann, San Francisco (1999)
29. Koza, J., Keane, M., Streeter, M., Mydlowec, W., Yu, J., Lanza, G.: Genetic Programming IV: Routine Human-Competitive Machine Intelligence. Kluwer Academic Publishers, Dordrecht (2003)
30. Lo, A.: The adaptive market hypothesis: market efficiency from an evolutionary perspective. Journal of Portfolio Management 30, 15–29 (2004)
31. Lo, A.: Reconciling efficient markets with behavioral finance: The adaptive markets hypothesis. Journal of Investment Consulting 2, 21–44 (2005)
32. Lux, T.: Herd behavior, bubbles and crashes. Economic Journal 105, 880–896 (1995)
33. Lux, T.: Time variation of second moments from a noise trader/infection model. Journal of Economic Dynamics and Control 22, 1–38 (1997)
34. Lux, T.: The socio-economic dynamics of speculative markets: Interacting agents, chaos and the fat tails of return distributions. Journal of Economic Behavior and Organization 33, 143–165 (1998)
35. MacQueen, J.B.: Some methods for classification and analysis of multivariate observations. In: Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, pp. 281–297. University of California Press, Berkeley (1967)
36. O'Hara, M.: Market Microstructure Theory. Blackwell, Oxford (1995)
37. O'Hara, M., Easley, D.A.: Liquidity and valuation in an uncertain world. Journal of Financial Economics 97(1), 1–11 (2010a)
38. O'Hara, M., Easley, D.A.: Microstructure and ambiguity. Journal of Finance 65(5), 1817–1846 (2010b)
39. Palmer, R., Arthur, W., Holland, J., LeBaron, B., Tayler, P.: Artificial economic life: a simple model of a stock market. Physica D 75, 264–274 (1994)
40. Phillips, P., Perron, P.: Testing for a unit root in time series regression. Biometric 75, 335–346 (1988)
41. Poli, R., Langdon, W., McPhee, N.: A Field Guide to Genetic Programming (2008), `http://Lulu.com`
42. Sansone, A., Garofalo, G.: Asset price dynamics in a financial market with heterogeneous trading strategies and time delays. Physica A 382, 247–257 (2007)
43. Simon, H.: Rational choice and the structure of environments. Psychological Review 63, 129–138 (1956)
44. Xu, R., Wunsch, D.: Clustering. Wiley-IEEE Press (2008)

Author Index

Subject Index